AMSTRAD

LE TOUR DE L'AMSTRAD

Pierre Raauenes-Gérard Sitbon



cedic/nathan

Le tour de l'Amstrad

Le tour de l'Amstrad

Pierre Raguenes
Gérard Sitbon

cedic/nathan
6-10 boulevard Jourdan, 75014 Paris - Tél. (1) 565-06-06

Éditions Cedic
6-10, boulevard Jourdan, 75014 - Paris
Téléphone : (1) 565-06-06

Illustrations : Véronique Manoukian
Maquette : Sophie Lunais

ISBN 2-7124-0594-3

Imprimé en France

Sommaire

Introduction

Lorsque l'on évoque l'Amstrad, on pense immédiatement à ses qualités les plus connues : qualité de la couleur et du graphisme, qualité des sons. Ces qualités, qui font le renom de ce micro-ordinateur, ne doivent pas faire oublier toutes les possibilités nouvelles qu'il offre : redéfinition graphique des caractères, fenêtres de texte, interruptions logicielles, gestion de la musique en temps réel, redéfinition du clavier, etc. Avec l'Amstrad, la micro-informatique familiale et semi-professionnelle entre dans l'ère de l'informatique tout court. Elle lui emprunte ses instruments les plus puissants, et devient ainsi plus simple à utiliser car plus performante.

Ce livre tente de faire un tour d'horizon des nouveautés introduites par l'Amstrad. La première partie est consacrée à l'étude des instructions du BASIC dans les domaines du graphisme, du son, des interruptions et du clavier. La seconde partie est une étude plus approfondie de l'Amstrad : organisation de la mémoire, initiation au langage machine, liste de sous-programmes "machine". Enfin, dans la troisième partie, nous vous présentons une série de programmes, jeux ou utilitaires spécifiques à l'Amstrad.

Chaque chapitre important est lui-même suivi d'un ou de deux programmes offrant un exemple concret d'utilisation des instructions abordées. Ces programmes vous aideront aussi à tirer le meilleur parti de votre Amstrad : éditeur de caractère, bibliothèques d'enveloppes de son et de tonalité, programme d'aide à la mise au point de programmes machine.

Chapitre I

Plein les yeux !

Trois décors : les modes d'affichage

Un ordinateur présente généralement un seul mode d'affichage. L'écran est divisé en **pixels**, parties élémentaires de l'image les plus petites que l'ordinateur puisse contrôler. L'ordinateur réserve dans sa mémoire une zone **mémoire écran** dans laquelle il mémorise, pour chaque pixel, la couleur de ce pixel.

C'est la taille de la mémoire réservée à l'écran qui détermine la qualité de l'image. Plus celle-ci est importante, plus un grand nombre de pixels peut être défini, et plus, pour chaque pixel, on peut choisir de couleurs.

L'utilisateur de l'Amstrad a le choix entre trois modes d'affichage. Il peut déterminer dans une certaine gamme la taille des pixels et le nombre de couleurs disponibles pour chaque pixel.

Les trois modes

Haute résolution : Mode 2

Dans ce mode, un pixel a la forme suivante :

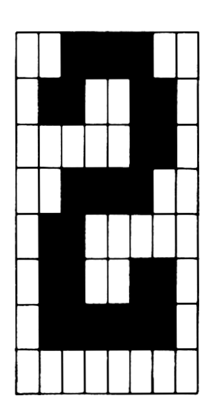
1 pas

2 pas

Le pixel est deux fois plus haut que large.
On peut le colorier en deux couleurs seulement.
L'écran est divisé en 128 ØØØ pixels, 64Ø en largeur, multipliés par 2ØØ en hauteur.

Dans ce mode, comme dans tous les modes, un caractère est représenté dans une **matrice** de 8 × 8 pixels.

Le chiffre 2 dans une matrice 8 × 8.



Dans ce mode, un caractère est deux fois plus haut que large.

En largeur on peut placer 64Ø/8 caractères, soit 8Ø caractères par ligne.

En hauteur, on peut placer 2ØØ/8 caractères, soit 25 lignes.

Si la taille de la mémoire écran de l'Amstrad est toujours la même, quel que soit le mode d'affichage choisi, elle est très simple à calculer en mode 2. On ne peut afficher un pixel qu'avec l'une des deux couleurs disponibles dans ce mode. Un bit suffit donc à représenter la couleur désirée (Ø pour la couleur 1, 1 pour la couleur 2). 128 ØØØ bits, c'est-à-dire 128 ØØØ/8 = 16 ØØØ octets, suffisent donc à représenter les 128 ØØØ pixels de l'écran.

Moyenne résolution : Mode 1

Dans ce mode, un pixel a la forme suivante :

2 pas

2 pas

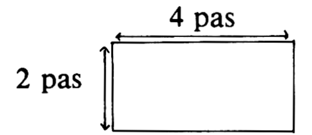
Il est aussi haut que large : quatre couleurs sont disponibles pour le colorier.

L'écran est divisé en 32Ø × 2ØØ = 64 ØØØ pixels. Il est large de 32Ø/8 = 4Ø caractères et haut de 2ØØ/8 = 25 caractères.

Deux bits sont nécessaires pour représenter le numéro de la couleur d'un pixel (Ø à 3 donc ØØ à 11). On retrouve les 128 ØØØ bits (64 ØØØ pixels × 2 bits) de la mémoire écran.

Basse résolution : Mode Ø

Dans ce mode le pixel est deux fois plus large que haut :



Seize couleurs sont disponibles pour le colorier.

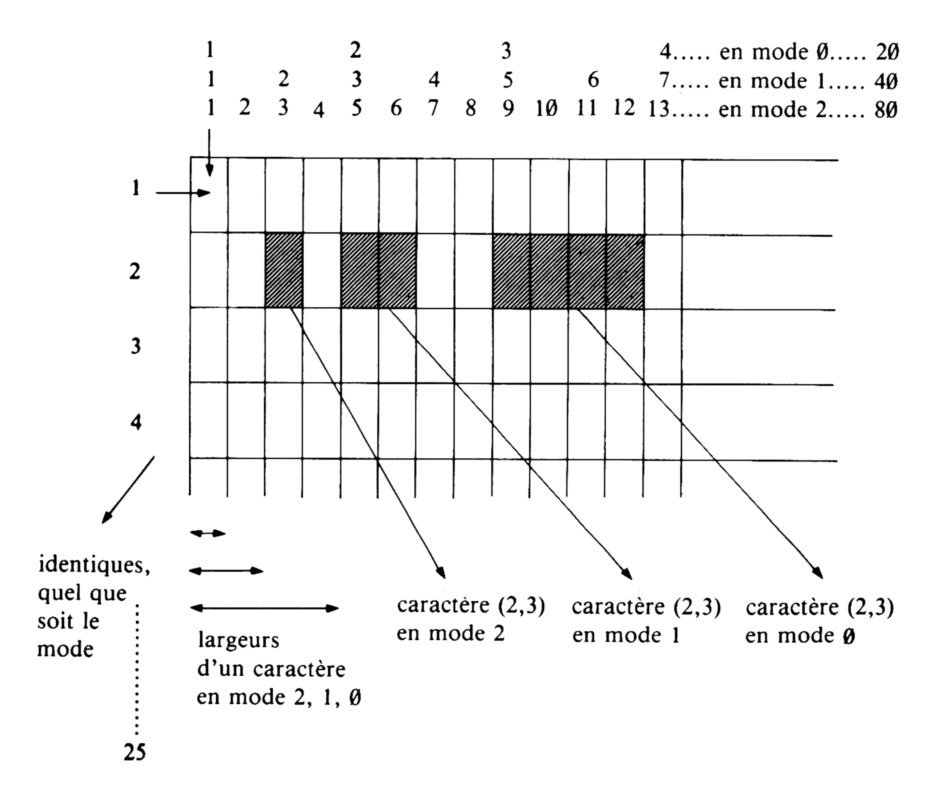
L'écran est divisé en 16Ø × 2ØØ = 32 ØØØ pixels. Il est large de 16Ø/8 = 2Ø caractères et haut de 2ØØ/8 = 25 caractères.

Quatre bits sont nécessaires pour représenter la couleur d'un pixel (Ø à 15 donc ØØØØ à 1111). La mémoire écran reste bien de 128 ØØØ bits = 32 ØØØ pixels × 4 bits.

Points de repère dans le texte

Il est habituel de situer un caractère sur l'écran en utilisant les **coordonnées de texte**. Celles-ci permettent de repérer un caractère à partir du caractère en haut à gauche de l'écran, en nombre d'emplacements de caractère. Comme la taille des caractères varie d'un mode à l'autre, les mêmes coordonnées ne désignent pas le même caractère suivant chaque mode.

La figure suivante montre les emplacements dans les différents modes d'un caractère de coordonnées texte (2, 3).



Le **curseur de texte** est l'emplacement caractère où doit être écrit le prochain caractère. Lorsque l'on affiche des caractères, il se déplace vers la droite et en fin de ligne, il saute à la ligne suivante. L'instruction LOCATE permet de positionner le curseur de texte. Elle est suivie des coordonnées de texte désignant le nouvel emplacement du curseur. (Nous verrons plus loin qu'en réalité, l'Amstrad possède 8 curseurs de texte affectés à 8 fenêtres.)

Repères dans les dessins

Les points graphiques

Contrairement aux coordonnées de texte, les **coordonnées graphiques** sont indépendantes du mode d'affichage. En effet, on ne repère

pas sur l'écran des pixels, mais des "points graphiques" fictifs de largeur et de hauteur égales.

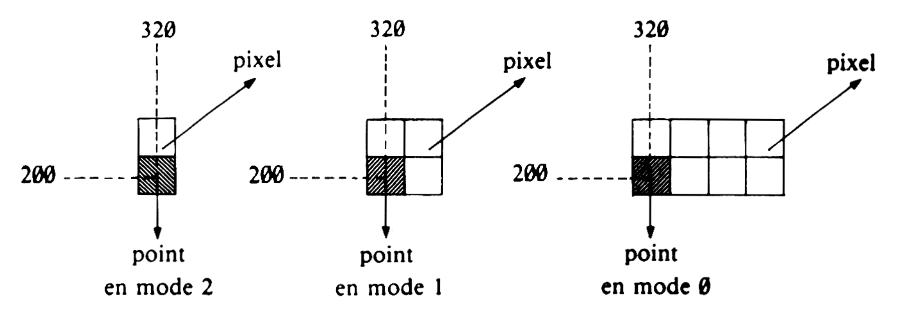
1 pas

1 pas ▢ Un point a la surface d'un demi-pixel de mode 2.

L'écran est ainsi divisé en une mosaïque de 640 × 400 points. N'en déduisez pas que la résolution de l'Amstrad est de 640 × 400 ; elle est bien au maximum de 640 × 200 en mode 2. Les concepteurs de l'Amstrad ont simplement choisi des coordonnées graphiques :

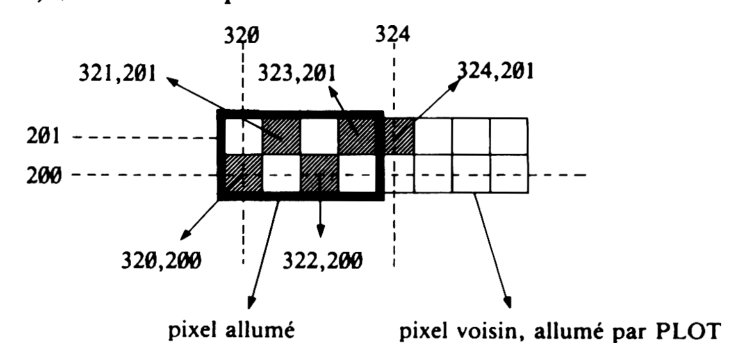
- indépendantes du mode d'affichage,
- telles que les longueurs des pas en largeur comme en hauteur soient égales.

Lorsque l'on "allume" un point, on allume en réalité **le pixel qui contient ce point**. Voici les pixels allumés lorsque l'on fait par exemple PLOT 320, 200 (allumer le point de coordonnées 320 et 200), dans les différents modes.



Pour vous en convaincre, passez en mode 0 et vérifiez que les instructions PLOT 320,200, PLOT 321,201, PLOT 322,200 et PLOT 323,201 ont pour effet d'allumer le même pixel.

PLOT 324,201 allume le pixel voisin.

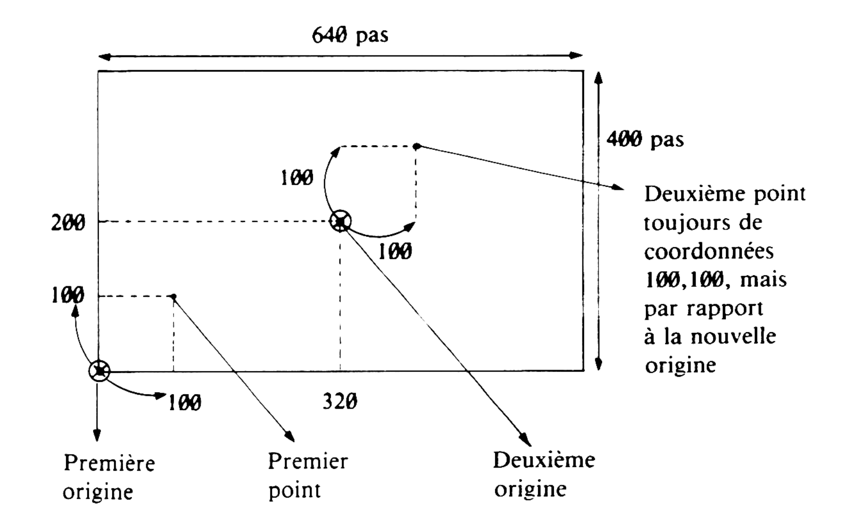


L'origine graphique

L'origine graphique est le point par rapport auquel sont repérés tous les autres points de l'écran. A la mise sous tension de l'Amstrad, celle-ci se trouve en bas à gauche de l'écran. Il est possible de modifier l'origine graphique par l'instruction ORIGIN.

Le programme suivant allume deux points de mêmes coordonnées mais par rapport à deux origines différentes.

```
10  ORIGIN 0,0
20  PLOT 100,100
30  ORIGIN 320,200
40  PLOT 100,100
```



Le curseur graphique

Le curseur graphique est le dernier point de l'écran repéré par une instruction graphique. Comme le curseur de texte pour les caractères, il "suit" les dessins effectués et peut être déplacé par l'instruction MOVE (équivalente du LOCATE).

Couleurs disponibles dans chaque mode

Oubliez l'instruction INK, les 27 couleurs (ou teintes de gris pour les moniteurs monochromes) de l'Amstrad, et supposez que l'Amstrad

ne possède que 16 couleurs placées dans 16 encriers. (Dans la suite, jusqu'à l'instruction INK, nous emploierons uniquement le terme d'**encrier** au lieu de **couleur**.)

Suivant le mode d'affichage choisi, le nombre d'encriers disponibles est différent. Si l'on choisit pour un pixel l'encrier n° 11 (1Ø11 en binaire), celui-ci sera ainsi mémorisé dans la mémoire écran.

[1Ø11] 11 dans le mode Ø
1Ø [11] 3 dans le mode 1
1Ø1 [1] 1 dans le mode 2

Seuls les bits de droite (de poids faibles) du numéro de l'encrier sont pris en compte. On retrouve bien le nombre de bits réservés à la couleur pour chaque pixel : 4 bits dans le mode Ø, 2 bits dans le mode 1, 1 bit dans le mode 2.

Ainsi, sont disponibles les encriers Ø et 1 dans le mode 2, les encriers Ø à 3 dans le mode 1, les encriers Ø à 15 dans le mode Ø.

La couleur

Colorier un caractère, un point ou une droite

Pour colorier les pixels, nous disposons toujours de nos 16 couleurs placées dans nos 16 encriers. Tant que l'on ne précise rien, les pixels sont coloriés avec :

- l'encrier 1, pour les pixels "allumés" des caractères et pour les graphiques (points ou droites),
- l'encrier Ø, pour les pixels "éteints" des caractères (le "papier").

Les instructions PEN et PAPER permettent de changer d'encrier pour écrire les caractères. De même, dans les instructions du type PLOT ou DRAW, on peut préciser, derrière les coordonnées, le numéro de l'encrier désiré.

Dans les deux cas (caractères et graphes) l'encrier fixé par ces instructions reste valable jusqu'à ce que l'on choisisse un autre encrier.

```
1Ø   PEN 1 : PAPER 12        ‘ Stylo encrier 1. Papier encrier 12.
2Ø   PRINT “BONJOUR”         ‘ Écrit avec encriers 1 et 12.
```

Tous les caractères qui sont affichés jusqu’à la nouvelle commande sont coloriés avec les encriers 1 et 12.

```
1ØØ  PEN 13 : PAPER Ø        ‘ A partir de maintenant, les caractè-
11Ø  PRINT “BONJOUR”           res sont écrits avec les encriers 13
                               et Ø.
12Ø  PLOT 32Ø,2ØØ, 3         ‘ Les graphes sont coloriés avec
                               l’encrier 3.
13Ø  DRAW 34Ø,2ØØ            ‘ Colorié avec 3.
```

Tous les graphes qui sont affichés sont coloriés avec l’encrier 3.

```
2ØØ  DRAW Ø,Ø, 4             ‘ A partir de maintenant, les graphes
21Ø  DRAW 639,39Ø            ‘ sont coloriés avec l’encrier 4.
```

Lorsque l’on change d’encrier, les caractères ou graphes **déjà** affichés ne changent pas de couleur. Ceci permet de colorier individuellement les caractères et les graphes.

Voici un programme de démonstration.

```
10 'caracteres
20 '
30 MODE 0: m=0: n=8
40 DATA c,o,u,l,e,u,r,s,        'caracteres a afficher
50 FOR i=0 TO 7
60 PAPER i+m: PEN i+n           'modification des encriers pour
70 READ a$: PRINT a$;: NEXT     'chaque caractere individuellement
80 PAPER 0: PEN 1               'restitue les anciens encriers
90 '
100 'graphiques
110 FOR i=0 TO 15
120 PLOT 0,5*i,i                'point + choix de l'encrier i
130 DRAW 639,5*i: NEXT          'la droite est coloriee avec l'encrier i
```

27 couleurs ou teintes de gris pour 16 encriers.

Bon nombre d’utilisateurs de micro-ordinateurs seraient déjà ravis de disposer de 16 couleurs et de pouvoir colorier à leur gré caractères, points et droites.

L’Amstrad, lui, propose 27 couleurs ou teintes de gris. C’est un avantage bien sûr, mais cela complique un petit peu l’utilisation des couleurs.

Pour bien la comprendre, nous vous proposons de reprendre notre métaphore des 16 encriers, numérotés de Ø à 15. Lorsque l'on fixe les couleurs des caractères par PEN et PAPER, des points ou des droites dans les instructions PLOT et DRAW, on ne donne pas un numéro de couleur, mais un numéro d'encrier. Les pixels sont coloriés avec **l'encre contenue dans cet encrier.**

Pour utiliser les 27 couleurs, on doit modifier le contenu des encriers (initialement remplis, à la mise sous tension, de 16 encres de couleurs différentes). L'instruction INK permet de "remplir" un encrier avec une nouvelle couleur. Tous les pixels allumés à cet instant avec cet encrier changent alors instantanément de couleur. Par exemple INK 3,19 qui remplit l'encrier n° 3 avec de l'encre de couleur n° 19 a pour effet de colorier tous les pixels allumés avec l'encrier 3 en couleur n° 19 (vert marin - gris très clair).

La liste des numéros de chaque couleur est donnée dans le manuel utilisateur.

Ø Noir	Ø Noir
1 Bleu	1 Gris très foncé
couleurs des plus foncées aux plus claires pour les moniteurs polychromes.	dégradé des gris foncés aux gris clairs pour les moniteurs monochromes.
25 Jaune pastel	25 Gris très clair
26 Blanc	26 Blanc

Voici un petit programme qui remplit dans les encriers Ø à 15, dans l'ordre, les encres de couleur noire, bleu vif, mauve, rouge, rouge vif, verte, bleu ciel, jaune, blanche, orange, rose, vert vif, vert citron, jaune vif, jaune pastel, blanc brillant.

```
5  'encriers 00,01,02,03,04,05,06,07,08,09,10,11,12,13,14,15
10 DATA      00,02,03,06,05,09,11,12,13,15,16,18,21,24,25,26
15 '
20 FOR i=0 TO 15: READ a: INK i,a: NEXT
```

La ligne DATA donne les numéros des couleurs choisies. La ligne 2Ø boucle en lisant ces données et en les plaçant dans les seize encriers.

Si l'on écrit PLOT 32Ø,2ØØ,11 dans chaque mode nous savons que l'encrier utilisé est, 11 dans le mode Ø, 3 dans le mode 1, 1 dans le mode 2.

PLOT 32Ø,2ØØ,11 affiche donc :

— en mode Ø, un point de couleur vert vif (n° 18) contenu dans l'encrier n° 11 ;
— en mode 1, un point de couleur rouge vif (n° 6) contenu dans l'encrier n° 3 ;
— en mode 2, un point de couleur bleu vif (n° 2) contenu dans l'encrier n° 1 ;

Plutôt que de chercher à deviner, comme nous venons de le faire, quelle couleur va "donner" telle ou telle instruction, nous vous conseillons d'adopter la démarche suivante : choisissez vos couleurs ; placez-les dans des encriers ; puis en fonction des encriers que vous avez utilisés, fixez pour le papier, le stylo et les dessins, les encriers voulus.

Si vous voulez par exemple, pour un programme en mode 1, écrire les caractères en rouge sur fond blanc et dessiner en bleu, votre programme pourra commencer ainsi.

```
10 'definition des couleurs
20 MODE 1
30 INK 0,13              'encrier 0 contient blanc
40 INK 1,3               'encrier 1 contient rouge
50 INK 2,1               'encrier 2 contient bleu
60 PAPER 0               'papier encrier 0 ---> blanc
70 PEN 1                 'stylo  encrier 1 ---> rouge
80 PLOT 0,0,2            'point  encrier 2 ---> bleu
90 '
```

La suite du programme vous permettra de vérifier que les couleurs affichées sont bien celles prévues.

```
91 '
100 'dessin d'une ellipse
110 '
120 'demande de la largeur et hauteur de l'ellipse
130 INPUT "largeur entre 1 et 319";larg
140 IF larg<1 OR larg>319 THEN GOTO 130
150 INPUT "hauteur entre 1 et 199";haut
160 IF haut<1 OR haut>199 THEN GOTO 150
170 '
180 'affichage des valeurs
190 CLS: PRINT "largeur="larg: PRINT "hauteur="haut
200 '
210 'trace de l'ellipse
220 ORIGIN 320,200: DEG
```

```
230 FOR i=0 TO 359 STEP 1
240 PLOT larg*COS(i),haut*SIN(i): NEXT
250 '
260 'suite ou non
270 PRINT "une touche SVP"
280 a$="": WHILE a$="": a$=INKEY$:WEND
290 IF a$<>"x" THEN CLS: GOTO 130: ELSE END
```

Ce programme trace au centre de l'écran une ellipse, dont il a demandé auparavant la largeur et la hauteur. Vous vérifierez que si les largeur et hauteur sont égales on obtient un cercle.

En fin de tracé de la courbe, frappez X pour sortir du programme, un autre caractère pour tracer une nouvelle ellipse.

Deux couleurs dans un encrier. Flash

On peut placer dans un encrier deux encres différentes.
INK 3,17,18 place alternativement dans l'encrier n° 3 les couleurs n° 17 et n° 18. A l'écran les pixels allumés par l'encrier 3 "flashent".

La vitesse de flash peut être réglée par l'instruction SPEED INK. SPEED INK 5Ø,1ØØ indique, en unités de 1/5Ø^{e} de seconde, les durées respectives durant lesquelles chaque encre reste dans l'encrier.

Dans notre cas, l'encrier 3 contiendra durant $5Ø \times \frac{1}{5Ø} = 1$ seconde la couleur 17, puis durant $1ØØ \times \frac{1}{5Ø} = 2$ secondes la couleur 18 et ainsi de suite.

Le "flash" peut être très intéressant pour afficher des caractères ou points clignotants sur lesquels on veut attirer l'attention. Il peut aussi être utilisé dans des applications psychédéliques très artistiques, mais attention aux yeux !

Il peut être tout aussi intéressant, sur programme d'interruption par exemple (voir plus loin), de redéfinir périodiquement le contenu des encriers et d'obtenir des effets moins violents que le flash. Le programme que nous vous présentons exécute une permutation des couleurs dans les douze encriers utilisés pour dessiner un cercle. Les couleurs choisies se trouvent dans la gamme 4 à 15. La boucle commençant à la ligne 17Ø exécute la permutation du contenu des encriers. Les lignes 22Ø et 23Ø "décalent" après chaque permutation la valeur initiale de la couleur. Ce programme a autant d'intérêt pour les moniteurs monochromes, que pour les polychromes.

```
10 ' impression de deplacement
20 '
30 ' remplir les encriers
40 MODE 0: INK 0,0: PAPER 0: DEG
50 FOR i=4 TO 15: INK i,i+10: NEXT
60 '
70 ' dessin d'un cercle avec 12 couleurs
80 FOR i=0 TO 359
90     ORIGIN 320,200
100    k=INT(i/15): k=k-12*INT(k/12)+4
110    DRAW 100*COS(i),100*SIN(i),k
120 NEXT
130 '
140 '64 permutations des 12 couleurs du cercle
150 coul=25
160 FOR nb=0 TO 63
170   FOR i=4 TO 15
180   INK i,coul
190   coul=coul+1
200   IF coul>25 THEN coul=coul-12
210   NEXT
220 coul=coul-1
230 IF coul<14 THEN coul=coul+12
240 NEXT
```

BORDER

Cette instruction n'est pas tout à fait équivalente à PEN ou PAPER. C'est un encrier supplémentaire dans lequel on place directement la (ou les 2) couleur(s).

Exemple :

BORDER 3,24 fait "flasher" le cadre de l'écran en rouge (couleur n° 3) et en jaune (couleur n° 24).

Modes d'écritures spéciaux

— Écriture de caractère.

Le caractère de contrôle 22 permet de modifier l'écriture des caractères.

PRINT CHR$(22) + CHR$(1) oblige les caractères à être écrits en mode transparent. Dans ce mode, seuls les pixels "allumés" des caractères sont pris en compte ; les pixels éteints ne sont pas affichés à l'écran. Ceci permet de faire cohabiter caractères et graphiques. PRINT CHR$(22) + CHR$(Ø) permet de revenir au mode normal, opaque.

— Affichage des graphiques.

Le caractère de contrôle 23 permet de modifier l'affichage des graphiques.

PRINT CHR$(23) + CHR$(Ø) commande le mode normal.
PRINT CHR$(23) + CHR$(1) commande le mode XOR.
PRINT CHR$(23) + CHR$(2) commande le mode AND.
PRINT CHR$(23) + CHR$(3) commande le mode OR.

Dans l'un des trois modes XOR, AND, OR un pixel graphique n'est pas affiché avec la couleur de son encrier (comme dans le mode normal), mais avec la couleur de l'encrier de numéro :

numéro de l'encrier du pixel actuellement à l'écran.	{ XOR AND OR }	numéro de l'encrier spécifié pour le pixel

Par exemple, en mode AND, si le pixel de coordonnées 32Ø,2ØØ est affiché avec l'encrier 13 (11Ø1) et si l'on fait PLOT 32Ø,2ØØ,11 (encrier 1Ø11), le pixel sera colorié avec l'encrier 9 (1ØØ1). En effet 11Ø1 AND 1Ø11 = 1ØØ1.

Ces commandes sont très pratiques, mais elles nécessitent un choix judicieux des encriers et des couleurs. Voici, nous l'espérons, deux exemples de choix judicieux !

```
10 MODE 0: DATA 0,15,6,24,: FOR i=0 TO 3: READ a: INK i,a: NEXT
20 PAPER 0: PEN 3: CLS: FOR i=0 TO 2: PRINT "bonjour": NEXT
30 FOR i=0 TO 2: PRINT CHR$(23)+CHR$(i+1);
40 PLOT 0,391-16*i,2: DRAW 223,391-16*i: NEXT
```

Les instructions 1Ø et 2Ø définissent les quatre couleurs utilisées et affichent trois fois "bonjour". L'instruction 4Ø raye les trois "bonjour" avec une droite dans les modes XOR, AND puis OR.

"bonjour" est entièrement rayé en mode XOR. Sur le fond en noir, 2 XOR Ø = 1Ø XOR ØØ = 1Ø = 2, la ligne apparaît en rouge (2 contient la couleur 6). Sur les caractères en jaune (encrier 3), 2 XOR 3 = 1Ø XOR 11 = Ø1 = 1, la ligne apparaît en orange (1 contient la couleur 15).

La droite apparaît "devant" le mot.

En mode AND, seuls les caractères sont rayés. En effet
2 AND Ø = 1Ø AND ØØ = ØØ = Ø,
la ligne n'apparaît pas sur le fond.
2 AND 3 = 1Ø AND 11 = 1Ø = 2,
la ligne est tracée en rouge sur le caractère.

En mode OR on a
2 OR Ø = 1Ø OR ØØ = 1Ø = 2,
la ligne apparaît sur le fond.
2 OR 3 = 1Ø OR 11 = 11 = 3,
la ligne n'apparaît pas sur le caractère.

La droite apparaît "derrière" le mot.

Le programme suivant dessine une spirale, en mode OR. Le papier est en encrier Ø (couleur bleue), l'intérieur de la spirale en encrier 2 (couleur rouge), les points de la spirale en encrier 3 (couleur jaune). Les encriers 2 et 3 sont toujours visibles sur le papier (2 OR Ø = 2, 3 OR Ø = 3). Les points de la spirale sont prépondérants sur l'intérieur (3 OR 2 = 3). Ceci permet de dessiner très facilement la spirale.

```
10 'dessin et coloriage d'une spirale
20 'initialisation encres, modes d'ecriture
30 MODE 1: INK 0,1: INK 1,3:INK 2,3: INK 3,24
40 PAPER 0: PEN 1               'papier en encrier 0
50 PRINT CHR$(23)+CHR$(3);      'mode OR
60 DEG: ORIGIN 320,200
70 '
80 'interieur de la spirale en encrier 2
90 'plus "fort" que 0 (papier), moins "fort" que 3
100 FOR i=0 TO 379 STEP 1
110 MOVE 0,0
120 DRAW (i/2)*COS(2*i),(i/2)*SIN(2*i),2
130 '
140 'coloriage des points de la spirale en encrier 3
150 'plus "fort" que 0 et 2
160 PLOTR 0,0,3: NEXT
```

Les instructions de texte
Les huit canaux d'affichage

Changeons nos habitudes

Les instructions de texte qu'offre le BASIC Amstrad sont CLS, LOCATE, PAPER, PEN, POS et VPOS, TAG et TAGOFF ainsi

que INPUT, LIST et PRINT. La plupart sont "standard". CLS : efface l'écran ; LOCATE : positionne le curseur sur l'écran ; PRINT affiche un caractère à l'écran... Nous avons déjà vu PEN et PAPER qui fixent les couleurs des caractères. Quatre instructions sont moins connues : POS, VPOS, TAG et TAGOFF; nous les verrons plus loin.

La particularité du BASIC Amstrad est qu'en réalité ces instructions ne "travaillent" pas sur l'écran, mais sur une **fenêtre**. Et si l'on veut continuer à parler d'**écran** pour la description des instructions de texte, il faut alors dire que l'Amstrad possède huit écrans ! Et ceci nous oblige à changer nos habitudes.

CLS n'efface pas l'écran.

CLS est en réalité CLS # Ø, qui efface la fenêtre Ø.

LOCATE 2,3 ne positionne pas le curseur de texte au point de coordonnées (2,3) de l'écran.

LOCATE 2,3 est en réalité LOCATE # Ø, 2, 3. Cette instruction positionne le curseur de l'écran de texte numéro Ø, au point de coordonnées (2,3) dans la fenêtre Ø.

Les huit canaux d'affichage

Le BASIC Amstrad gère en effet huit canaux d'affichage. Il offre ainsi la possibilité d'**acheminer** un caractère jusqu'à l'écran "physique" du moniteur, à travers l'un de ces huit canaux. A chaque canal correspond une zone bien déterminée de l'écran : une fenêtre. Une fenêtre peut occuper tout ou une partie de l'écran. Les fenêtres peuvent se superposer.

Voici les caractéristiques que l'on peut affecter à chaque canal, indépendamment les uns des autres :

- la taille et la position de la fenêtre sur l'écran,
- la position du curseur affecté à ce canal, dans la fenêtre,
- le choix des encriers pour le papier (PAPER) et le stylo (PEN) de ce canal,
- l'écriture en mode opaque ou transparent dans la fenêtre,
- l'écriture des caractères en mode graphique (voir TAG).

En revanche, les caractéristiques suivantes ne peuvent être choisies indépendamment pour chaque canal :

- le mode d'affichage (MODE),
- la couleur dans les encriers (INK),
- les matrices des caractères (voir plus loin).

Derrière une instruction de texte, on doit préciser, si l'écran sur lequel on veut travailler n'est pas l'écran Ø, le numéro du canal voulu, précédé du signe #. Par exemple :

CLS #1 efface la fenêtre (l'écran) n° 1.

LOCATE #1, 2, 3 positionne le curseur n° 1 au point (2,3) de la fenêtre 1, etc.

A défaut, si le numéro de canal n'est pas indiqué, le BASIC exécute l'instruction sur la fenêtre Ø.

L'instruction WINDOW

Cette instruction permet de délimiter la fenêtre affectée à un canal. On indique en coordonnées de texte dans l'écran les limites gauche, droite, haute et basse de la fenêtre.

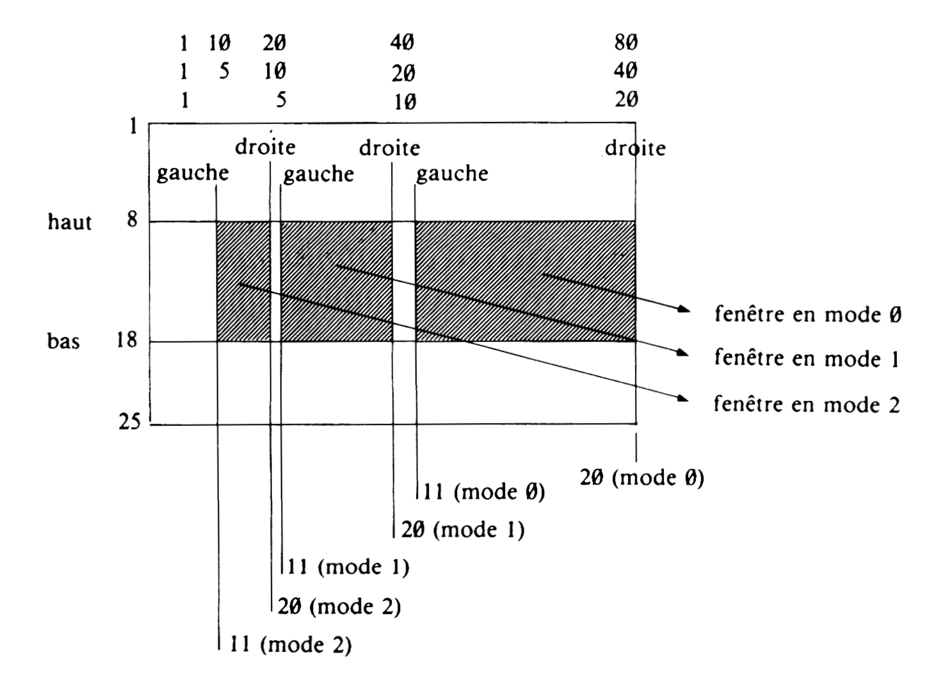
WINDOW # n° de canal, gauche, droite, haut, bas.

Exemple :

WINDOW #1, 11, 2Ø, 8, 18 affecte l'une de ces zones écran au canal 1.

Comme on le voit sur la figure suivante, la taille et l'emplacement de la fenêtre dépendent du mode d'affichage. Un programme qui utilise des fenêtres doit être entièrement conçu en fonction du mode d'affichage utilisé.

A la mise sous tension de l'Amstrad, les huit fenêtres affectées aux huit canaux occupent l'écran entier, quel que soit le mode d'affichage choisi, jusqu'à ce que l'on utilise WINDOW.



L'instruction WINDOW SWAP permet d'échanger entre deux canaux les limites des fenêtres allouées à chaque canal.

WINDOW SWAP 1,2 : la fenêtre du canal 1 devient celle du canal 2 et vice versa.

Intérêt des fenêtres

La technique des fenêtres présente un grand intérêt. Elle permet au programmeur de présenter les résultats de ses programmes, de questionner l'utilisateur d'une manière claire et agréable, sans difficulté de programmation particulière.

Le programme suivant divise l'écran en huit fenêtres, dont les limites sont calculées au sous-programme placé à la ligne 1000. Il rappelle la liste des instructions les plus importantes "travaillant" sur des fenêtres.

```
10 MODE 0
20 FOR i=0 TO 7                                  'huit fenetres
30    GOSUB 1000                                 'calcul des limites
35    WINDOW #i,gauche,droite,haut,bas           'declaration de la fenetre
40    PAPER #i,2*i                               'papier de la fenetre
50    PEN #i,2*i+1                               'stylo de la fenetre
60    CLS #i                                     'efface la fenetre
70    LOCATE #i,2,6                              'positionne dans la fenetre
80    PRINT #i,"abc"                             'affichage dans la fenetre
90 NEXT
100 END
998  '
999  'calcul des limites de la fenetre i
1000 gauche = (i-4*INT(i/4))*5+1
1010 droite = gauche + 4
1020 haut = (INT(i/4))*13+1
1030 bas  = haut + 12
1040 RETURN
```

Ce programme n'est qu'une illustration de l'utilisation des fenêtres. Il faut remarquer qu'il serait bien difficile d'obtenir le même résultat sans l'instruction WINDOW et les instructions de texte "par canal". La technique des fenêtres devient vite indispensable, dès que l'on en dispose.

Les instructions POS et VPOS

Ces instructions donnent la position dans une fenêtre du curseur de texte associé à la fenêtre. POS donne la coordonnée horizontale, VPOS la coordonnée verticale. Ces instructions peuvent être très utiles dans un programme de traitement de texte. Le programme suivant utilise POS pour afficher une ligne de prénoms sans les "couper" en fin de ligne, dans la mesure, bien sûr, où ceux-ci ne sont pas plus longs que les lignes. Essayez-le plusieurs fois en modifiant les prénoms dans les lignes DATA.

```
8 '
9 'la fenetre 0 occupe la moitie de l'ecran (20 caracteres)
10 MODE 1: WINDOW 1,20,1,25
18 '
19 'declaration des prenoms sur deux lignes DATA
20 DATA pierre,jean,luc,francois,albert
30 DATA elisabeth,paula,francine,ernestine,genevieve,
38 '
39 'pour les 10 prenoms, lire le prenom
40 FOR i=0 TO 9
50    READ a$
59 '    si le prenom a une longueur superieure a la fenetre
60      IF LEN(a$)>20 THEN GOTO 70 ELSE GOTO 80
68 '    alors si le curseur n'est pas en debut de ligne
69 '       alors retour a la ligne
```

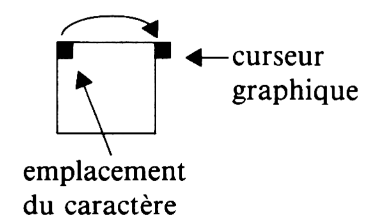
```
70        IF POS(#0)=1 THEN GOTO 100 ELSE PRINT: GOTO 100
77 '    sinon longueur restante est calculee dans long
78 '      si longueur du caractere est superieure a long
79 '      alors retour a la ligne
80      long=20-POS(#0)
90         IF LEN(a$)<long THEN GOTO 100 ELSE PRINT: GOTO 100
99 '   ecriture du prenom
100    PRINT a$" ";
110 NEXT
```

L'instruction POS permet de savoir, en 7Ø, si le curseur se trouve en début de ligne. Elle permet de calculer, en 8Ø, le nombre de caractères qu'il est encore possible d'écrire dans la ligne. Nous avons utilisé la fenêtre Ø, ramenée à la moitié de l'écran.

Les instructions TAG et TAGOFF

L'instruction TAG est l'une des instructions les plus puissantes que propose le BASIC Amstrad dans le domaine graphique. Elle permet, jusqu'à ce que l'on utilise TAGOFF d'écrire des caractères en mode graphique :

— les caractères sont écrits à l'emplacement du curseur graphique;



Les caractères sont affichés à l'emplacement pointé en haut à gauche par le curseur graphique.
Lorsque l'on écrit un caractère, le curseur graphique se déplace.

— les caractères sont coloriés avec la couleur **alors** utilisée par les graphiques pour les pixels "allumés" ; avec la couleur de l'encrier Ø pour les pixels "éteints".

On le voit, cette instruction permet d'afficher les caractères en dehors de la grille qui leur est normalement imposée (les emplacements de caractères). Elle permet donc d'afficher des caractères "un peu partout, dans tous les sens". Elle permet aussi de commenter sans difficulté un graphique, quelle que soit sa position à l'écran ; indication de l'équation de la courbe que l'on vient de tracer ; indica-

tion du contenu de chaque part dans une représentation en "camembert", etc. Nous vous donnerons des exemples d'utilisation de TAG dans les "instructions graphiques".

Les instructions graphiques

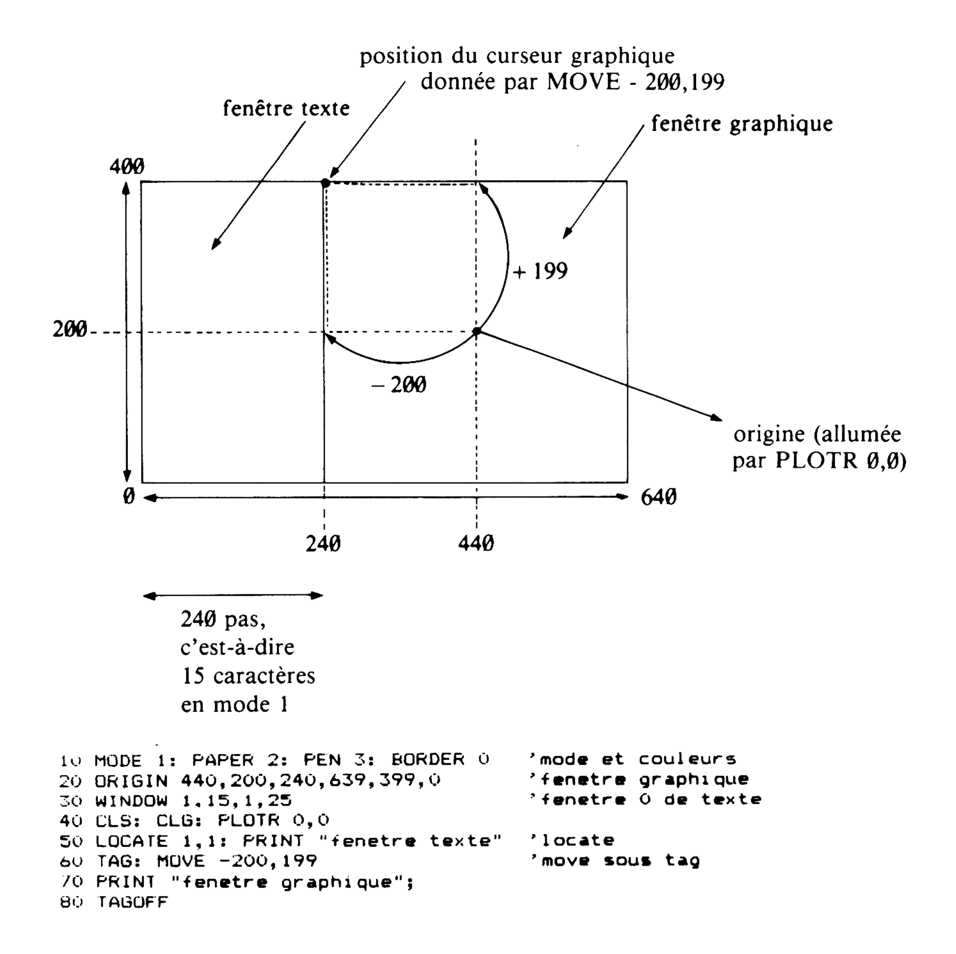
L'instruction ORIGIN. La fenêtre graphique. CLG

Nous avons déjà vu l'instruction ORIGIN. Elle permet de modifier l'emplacement du point par rapport auquel tous les autres points de l'écran seront repérés. Cette instruction peut être complétée par quatre paramètres supplémentaires donnant les limites gauche, droite, haute et basse de la **fenêtre graphique**. Si ces paramètres ne sont pas donnés, la fenêtre graphique n'est pas modifiée. Elle occupe, à la mise sous tension de l'Amstrad, l'écran entier.

ORIGIN coord. larg., coord. haut., gauche, droite, haut, bas.

Ces valeurs sont indiquées en coordonnées graphiques par rapport au point en bas à gauche de l'écran.

Le programme suivant définit une fenêtre graphique à droite de l'écran de 4ØØ pas de largeur sur 4ØØ pas de hauteur. Nous avons limité la fenêtre Ø de texte à la zone de l'écran restant. (Il reste en largeur 24Ø pas, soit 12Ø pixels en mode 1, soit 12Ø/8 = 15 caractères.) Nous utilisons l'instruction CLG qui permet d'effacer avec l'encrier Ø la fenêtre graphique. PLOTR Ø,Ø donne la position du curseur graphique, donc de l'origine à ce moment.
Nous avons écrit en haut à gauche de chaque fenêtre, en utilisant LOCATE pour positionner le curseur de texte, MOVE pour positionner le curseur graphique, le type de chaque fenêtre. Remarquez l'utilisation de TAG qui permet d'écrire dans la fenêtre graphique. Remarquez aussi, après avoir lancé ce programme, les couleurs de chaque zone. (Réinitialisez éventuellement votre Amstrad si vous avez beaucoup utilisé INK jusqu'à maintenant. Nous n'avons pas pris le soin de redéfinir les encriers.)




```
10 MODE 1: PAPER 2: PEN 3: BORDER 0         'mode et couleurs
20 ORIGIN 440,200,240,639,399,0             'fenetre graphique
30 WINDOW 1,15,1,25                         'fenetre 0 de texte
40 CLS: CLG: PLOTR 0,0
50 LOCATE 1,1: PRINT "fenetre texte"        'locate
60 TAG: MOVE -200,199                       'move sous tag
70 PRINT "fenetre graphique";
80 TAGOFF
```

L'intérêt de la fenêtre graphique est de pouvoir délimiter la zone de l'écran réservée aux graphiques. En dehors de cette zone, si l'on vient à "déborder", les dessins (et les écritures sous graphique) ne sont pas affichés. Avec l'instruction WINDOW elle permet de séparer très facilement, comme nous venons de le faire, l'écran en une zone de texte et une zone de dessin.

Remarque

Il est possible de donner à une coordonnée graphique une valeur allant de −32768 à +32767 (−&8ØØØ à +&7FFF). Rien n'empêche de placer l'origine ou un point en dehors de la fenêtre graphique et de l'écran. Frappez ces instructions pour vous en convaincre, à la suite de notre programme précédent.

ORIGIN -3ØØØØ, -3ØØØØ (l'origine se trouve bien loin de l'écran !).

PLOT 3Ø4ØØ, 3Ø2ØØ (le point apparaît sur l'écran).

DRAW 3Ø8ØØ, 3Ø2ØØ (la droite sort de la fenêtre graphique et de l'écran).

Les instructions graphiques permettent de "travailler" sur un immense plan de 65 536 × 65 536 pas dont la fenêtre graphique n'est que la minuscule partie visible.

Instructions graphiques

La liste des instructions graphiques de l'Amstrad est la suivante :

MOVE et MOVER déplacent le curseur graphique.

PLOT et PLOTR allument un point.

DRAW et DRAWR tracent une droite.

TEST et TESTR donnent la couleur (le n° de l'encrier) d'un point.

XPOS et YPOS fournissent les coordonnées horizontale et verticale du curseur graphique par rapport au point d'origine.

Coordonnées absolues et relatives

Nous reviendrons plus loin sur les instructions XPOS et YPOS. Les autres instructions graphiques sont toujours suivies des coordonnées d'un point(et d'un seul).

MOVE(R), le point où l'on place le curseur graphique.

PLOT(R), le point que l'on allume.

DRAW(R), le point de la seconde extrémité de la droite, la première extrémité étant le point où se trouve le curseur graphique.

TEST(R), le point dont on désire connaître la couleur.

(Éventuellement, comme nous l'avons vu, PLOT(R) et DRAW(R) peuvent encore être suivies du numéro de l'encrier utilisé pour la couleur.)

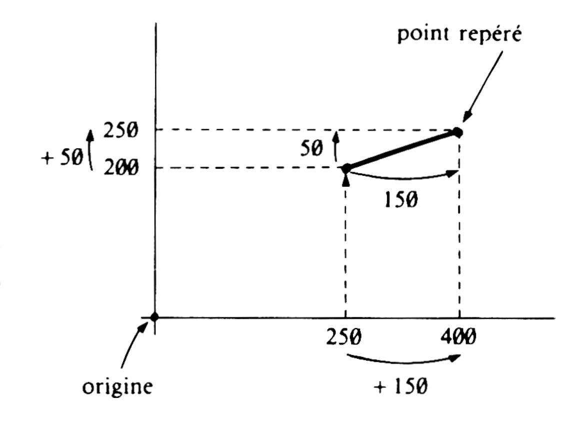
Ce qui différencie les instructions se terminant par "R" des autres instructions est la manière dont on indique l'emplacement du point. Dans les instructions du type MOVE, on donne les coordonnées du point **par rapport à l'origine**. (Spécifié dans la commande ORIGIN.) Dans les instructions du type MOVER, on donne les coordonnées du point **par rapport au curseur graphique**.

Rappelons que le curseur graphique est déplacé :

- après MOVE(R) vers le point spécifié,
- après PLOT(R) vers le point allumé,
- après DRAW(R) vers le point extrême de la droite tracée,
- après TEST(R) vers le point testé.
- lorsque l'on écrit sous graphique (voir TAG).

Le schéma suivant doit vous permettre de bien différencier les deux manières de "pointer" l'écran.

Si le curseur se trouve au point de coordonnées 25Ø et 2ØØ par rapport à l'origine, le point repéré, de coordonnées 4ØØ,25Ø par rapport à l'origine, a pour coordonnées par rapport au curseur 15Ø et 5Ø.



Dans ce cas les instructions PLOT 4ØØ,25Ø et PLOTR 15Ø,5Ø ont le même effet : allumer le point repéré. De même les instructions DRAW 4ØØ,25Ø et DRAWR 15Ø,5Ø ont le même effet : tracer la droite entre le curseur et le point repéré. Dans tous les cas, le curseur se trouve ramené au point repéré.

Le programme suivant donne un exemple d'utilisation des coordonnées absolues et relatives. L'instruction PLOT est utilisée pour **positionner** et dessiner une sinusoïde. L'instruction PLOTR permet de recopier cette courbe en diagonale 14 fois de suite, en 14 couleurs différentes. Chaque courbe est recopiée 1Ø pas à droite, 1Ø pas en haut, **relativement** à la précédente.

L'origine est placée en Ø,1ØØ. La boucle commence à dessiner un point de coordonnée horizontale – 14Ø, donc en dehors de l'écran (mais dans l'immense plan invisible). Ceci permet à la quatorzième recopie (et aux précédentes) d'être visible sur toute la largeur de l'écran.

```
10 MODE 0: CLS: DEG: ORIGIN 0,100,0,639,399,0
20 FOR i=-140 TO 639 STEP 1 'pour plus que largeur ecran
30     PLOT i,100*SIN(i),1   'tracer la sinusoide
40     FOR j=0 TO 13         'recopier 14 fois ce point
50         PLOTR 10,10,j+2   'aux coordonnees relatives 10,10
60 NEXT: NEXT
```

D'une manière générale, les instructions en coordonnées absolues sont plutôt utiles pour tracer des courbes à partir d'équations mathématiques bien déterminées. Les instructions relatives permettent de se déplacer "au coup par coup" dans une zone de l'écran dont on ne connaît pas, a priori, l'emplacement. Elles permettent de souligner très aisément des mots écrits sous mode graphique, de tracer des flèches vers un point bien précis d'un graphe, sur lequel on désire attirer l'attention, etc.

Dans le programme suivant, l'instruction MOVE permet de positionner le début des lignes que l'on écrit sous mode graphique (TAG). C'est ensuite l'instruction MOVER qui est utilisée pour diriger le curseur graphique, évitant ainsi des calculs compliqués et rendant le programme **indépendant du mode d'affichage**. (C'est pourquoi le programme vous demande le mode choisi — faites bien attention à répondre Ø, 1 ou 2, la réponse n'est pas contrôlée — faites ESC deux fois pour sortir du programme.)

```
10 DATA c,a, ,m,o,n,t,e, ,e
20 DATA t, ,c,a, ,d,e,s,c,e,n,d
30 DATA e,n, ,m,e,r, ,c,a, ,o,n,d,u,l,e,!
40 INPUT "mode";i: MODE i: ORIGIN 0,0: DEG
45 PLOT -1,-1,1  'initialise encrier des graphiques hors ecran
50 TAG
60 MOVE 50,360: PRINT "en montagne";
70 MOVE 0,200
80 FOR i=0 TO 9: MOVER 1,8: GOSUB 1000: NEXT   'ca monte
90 FOR i=0 TO 11: MOVER 1,-8: GOSUB 1000: NEXT 'ca descend
100 MOVE 0,100
110 FOR i=0 TO 16
120 MOVER 3,8*SIN(30*i): GOSUB 1000: NEXT      'ca ondule
130 TAGOFF: RESTORE: GOTO 40
998 '
999 'lecture et affichage de caractere
1000 READ a$: PRINT a$;: RETURN
```

Les instructions TEST et TESTR

Ces instructions permettent de connaître le numéro de l'encrier utilisé pour colorier un point, dont les coordonnées sont indiquées d'une manière absolue (TEST) ou relative (TESTR). Ces coordonnées sont placées entre parenthèses.

A = TEST (32Ø,2ØØ) place dans la variable A le numéro de l'encrier utilisé pour dessiner le point 32Ø,2ØØ.

Ces instructions s'avèrent bien utiles dans la panoplie du BASIC Amstrad. En effet la structure assez complexe de la mémoire écran (due aux trois modes d'affichage) interdit les analyses (et modifications) aisées de la mémoire écran par les traditionnels PEEK et POKE.

Nous vous proposons un exemple d'utilisation de TEST dans le programme suivant. Celui-ci est une variante de notre précédent programme traçant quatorze sinusoïdes. Dans ce programme on n'allume un point (PLOT ou PLOTR) que si ce point a la couleur du fond ; ici la couleur Ø. (Voir le sous-programme à la ligne 1ØØØ.) La modification a pour effet d'inverser l'impression de relief. Remarquez comment le programme initial a été modifié, et en particulier l'utilisation des coordonnées relatives (TESTR, PLOTR).

```
10 MODE 0: CLS: DEG: ORIGIN 0,100,0,639,399,0
20 FOR i=-140 TO 639 STEP 1
30     MOVE i,100*SIN(i): coul=1: GOSUB 1000
40     FOR j=0 TO 13
50         MOVER 10,10: coul =j+2: GOSUB 1000
60 NEXT:NEXT:END
998 '
999 'ecrit point si couleur actuelle est celle du fond
1000 IF TESTR(0,0)=0 THEN PLOTR 0,0,coul
1010 RETURN
```

Les instructions TEST permettent de dessiner "en fonction du terrain" ; de ne pas effacer telle ou telle partie de l'écran avec un graphe, etc. Elles peuvent aussi être utilisées pour une analyse fine de l'écran et en particulier des caractères. Nous en donnons un exemple dans la "redéfinition graphique des caractères".

Les instructions XPOS et YPOS

Ces instructions donnent les coordonnées horizontale (XPOS) et verticale (YPOS) du curseur graphique par rapport à l'écran.

a = XPOS : b = YPOS placent dans les variables a et b les coordonnées du curseur graphique.

XPOS et YPOS sont équivalentes pour le graphisme à POS et VPOS pour le texte. En particulier lorsque l'on écrit sous mode graphique elles permettent de savoir à quel moment il est nécessaire de revenir à la ligne (par un MOVE) pour éviter de "sortir" de la fenêtre graphique ou d'une zone choisie. Elles permettent aussi de mémoriser les coordonnées de points "stratégiques" vers lesquels on veut pouvoir revenir. Elles sont très utiles pour "se repérer" sur l'écran après avoir utilisé des instructions graphiques en coordonnées relatives.

La redéfinition graphique des caractères

L'Amstrad offre la possibilité de redéfinir le graphisme des caractères. Ceci permet à l'utilisateur (qui utilisera aussi la redéfinition des touches, voir le chapitre IV) d'adopter des alphabets différents, ou de compléter l'alphabet anglo-saxon de l'Amstrad. Il est ainsi possible, grâce à la redéfinition graphique, de dessiner les ç, à, ù, ï, ê, é, è... retrouvant ainsi les accents et cédille chers à notre vieille langue française, puis grâce à la redéfinition des touches de créer un clavier AZERTY, cher à notre cœur, et à nos habitudes !

Représentation d'un caractère

L'Amstrad contient en mémoire morte la forme initiale des 256 caractères qu'il possède. (Cette forme est donnée pour chaque caractère dans le manuel utilisateur.) La représentation en mémoire d'un caractère est extrêmement simple. Nous vous proposons de l'étudier pour le caractère 2.

matrice	binaire	hexadécimal	décimal
	Ø Ø 1 1 1 1 Ø Ø	&3C	6Ø
	Ø 1 1 Ø Ø 1 1 Ø	&66	1Ø2
	Ø Ø Ø Ø Ø 1 1 Ø	&Ø6	Ø6
	Ø Ø 1 1 1 1 Ø Ø	&3C	6Ø
	Ø 1 1 Ø Ø Ø Ø Ø	&6Ø	96
	Ø 1 1 Ø Ø 1 1 Ø	&66	1Ø6
	Ø 1 1 1 1 1 1 Ø	&7E	126
	Ø Ø Ø Ø Ø Ø Ø Ø	&ØØ	ØØ

A gauche nous avons représenté le caractère dans sa matrice de 8 × 8 pixels. Au centre nous avons remplacé les pixels allumés du caractère par un 1, les pixels éteints par un Ø. (On retrouve la forme du caractère en entourant la zone où se trouvent les 1.) Chacune des huit lignes du caractère est ainsi représentée par une suite de huit chiffres binaires (Ø ou 1), c'est-à-dire par un octet. Nous avons donné à droite la valeur en hexadécimal puis en décimal des huit octets représentant le caractère 2.

Les lecteurs non familiers avec les systèmes de numérotation binaire et hexadécimal pourront utiliser la méthode de calcul suivante.

On donne à chaque pixel d'une ligne une valeur : 128 pour le pixel le plus à gauche de la ligne, puis 64, 32, 16, 8, 4, 2 et 1 pour le pixel le plus à droite. La valeur d'une ligne est la somme des valeurs des pixels allumés. Par exemple la deuxième ligne du caractère 2, a pour valeur 64 + 32 + 4 + 2 = 1Ø2.

SYMBOL AFTER

Il faut, nous venons de le voir, huit octets pour représenter la forme d'un caractère. Pour 256 caractères, il faut donc 8 × 256 soit 2Ø48 octets, 2 k-octets de mémoire.

Ces 2 k-octets se trouvent bien sûr en ROM (car sinon les caractères auraient une forme bien bizarre à la mise sous tension de l'Amstrad). Par contre lorsque l'on modifie la forme d'un caractère, la représentation de celui-ci est obligatoirement stockée en RAM (la seule mémoire où l'on puisse écrire). L'instruction SYMBOL AFTER per-

met de réserver une zone de mémoire RAM qui sera utilisée pour stocker les nouvelles représentations de caractères. Cette instruction est suivie d'une valeur de Ø à 255. Elle signifie que tous les caractères dont la valeur ASCII est supérieure ou égale à la valeur indiquée ont leur représentation stockée en RAM, et qu'il est possible de les modifier. (Vous trouverez de plus amples informations sur les valeurs ASCII au chapitre IV.)

SYMBOL AFTER a pour effet de recopier **toutes** les représentations de caractère dont la valeur ASCII est supérieure ou égale à la valeur indiquée, de la zone ROM, où elles sont initialement stockées, vers la zone RAM réservée. Relisez bien ; cela signifie que les modifications de graphisme de caractères que vous auriez pu faire avant SYMBOL AFTER sont effacées. Cette remarque, ainsi que celle que vous trouverez dans l'étude du plan mémoire de l'Amstrad, doit achever de vous convaincre du principe suivant. **N'utiliser SYMBOL AFTER qu'une seule fois dans un programme** (et au début de celui-ci de préférence !).

L'instruction SYMBOL

Cette instruction permet de redéfinir le graphisme d'un caractère dont on donne la valeur ASCII, puis les huit octets de la représentation du caractère. La liste des valeurs ASCII de chaque caractère est fournie dans le manuel utilisateur. Pour être exécutée il faut que la valeur ASCII du caractère soit supérieure à celle autorisée par SYMBOL AFTER. Le plus simple est de commencer le programme par SYMBOL AFTER Ø.

Le programme suivant donne un exemple de modification du caractère 2. Sa valeur ASCII est 5Ø. Nous avons fait suivre l'instruction SYMBOL de cette valeur, puis des huit valeurs de la nouvelle représentation du caractère.

```
10 MODE 0: SYMBOL AFTER 0
20 PRINT "avant": PRINT: PRINT "  2  "
30 SYMBOL 50,126,98,2,126,64,70,126,0
40 LOCATE 7,1: PRINT "apres": LOCATE 9,3: PRINT "2": PRINT
```

Exécutez le programme deux fois pour vérifier que la seconde fois SYMBOL AFTER Ø a détruit la modification du caractère 2. Faites ensuite SYMBOL AFTER Ø pour retrouver un "2" normal.

Éditeur de caractères

Le programme suivant est un éditeur de caractères. Il vous demande tout d'abord si c'est la première fois que vous lancez ce programme. Répondez O pour Oui, N pour Non. Si vous répondez Non, le programme n'exécute pas l'instruction SYMBOL AFTER Ø, et n'efface donc pas les modifications de caractères que vous avez déjà effectuées. Il vous demande ensuite si vous voulez modifier un caractère. Si vous répondez N pour Non, le programme se termine. Si vous répondez par un autre caractère (par exemple O pour Oui) on vous demande la valeur ASCII (entre Ø et 255) du caractère que vous voulez modifier. (Répondez par exemple 5Ø qui est la valeur ASCII du caractère 2, pour une première utilisation.) Le programme affiche alors à droite de l'écran le caractère qu'il recopie dans une matrice plus visible. Un curseur clignotant est affiché. Pour le déplacer, utilisez les touches de déplacement du curseur. Pour modifier un pixel, placez le curseur sur celui-ci et frappez A pour l'allumer, E pour l'effacer. Lorsque le caractère a la forme voulue, faites O (pour OK). Le programme, après s'être assuré que vous ne vous êtes pas trompés (répondez O à "c'est sûr ?"), calcule les paramètres de l'instruction SYMBOL, qu'il affiche en haut à droite de l'écran, puis il affiche à droite le caractère redessiné.

Ce programme nous semble présenter un double intérêt. Tout d'abord sa fonction pourra vous être utile. Notez les résultats qui vous semblent intéressants, pour les utiliser en début d'un programme nécessitant une redéfinition des caractères. Ceci vous évitera d'avoir à les calculer vous-mêmes. Ensuite, nous avons tenté d'utiliser le plus possible d'instructions graphiques : WINDOW, ORIGIN, INK, PAPER, PEN, PLOT(R), DRAWR, TEST, MOVE(R), XPOS et YPOS, SYMBOL AFTER et SYMBOL.

```
1 '--------------------------------
2 '      editeur de caracteres
3 '--------------------------------
4 '
5 GOSUB 5000                  'initialisation
10 INPUT "autre caractere";a$: IF a$="n" THEN END
20 INPUT "valeur ASCII";ascii: IF ascii>255 THEN GOTO 20
29 '
30 GOSUB 3500                 'affiche le caractere
35 GOSUB 4000                 'le recopie dans la matrice
40 xc=0: yc=0                 'xc yc coordonnees curseur cligotant
```

```
45 GOSUB 1000              'allume le curseur en xc,yc
46 '
47 ' traitement des commandes
48 ' 240 est la fleche vers haut,241 est la fleche vers bas
49 ' 242 est la fleche vers gauche, 243 est la fleche vers gauche
50 ' sous-programme 1500 efface curseur clignotant
51 '                        ''
52 a$=INKEY$
60 IF a$=CHR$(240) THEN GOSUB 1500: yc=yc+1: GOSUB 1000
70 IF a$=CHR$(241) THEN GOSUB 1500: yc=yc-1: GOSUB 1000
80 IF a$=CHR$(242) THEN GOSUB 1500: xc=xc-1: GOSUB 1000
90 IF a$=CHR$(243) THEN GOSUB 1500: xc=xc+1: GOSUB 1000
91 '
92 ' si effacement colorier le carre pointe pas le curseur
93 ' avec la couleur 0, puis reallumer le curseur. idem si
94 ' allumage, mais avec couleur 2
95 '
99 '
100 IF a$="e" THEN cm=0: xm=xc: ym=yc: GOSUB 4500: GOSUB 1000
110 IF a$="a" THEN cm=2: xm=xc: ym=yc: GOSUB 4500: GOSUB 1000
120 IF a$<>"o" THEN GOTO 50
121 '
122 ' le caractere a ete valide
123 '
130 INPUT "c'est sur";a$          'verifier que c'est sur
135 IF a$<>"o" THEN GOTO 50
140 GOSUB 1500                     'effacer le curseur
145 FOR i=0 TO 7:psb(i)=0:NEXT 'reinitialiser parametres SYMBOL
146 '
147 ' calcul des parametres du caractere
148 '
150 MOVE 75,75                     'positionne haut droit matrice
151 FOR i=0 TO 7                   'de haut en bas
152     FOR j=0 TO 7               'de droite a gauche
160         IF TESTR(0,0)=2 THEN psb(i)=psb(i)+2^j
170         MOVER -10,0            'vers gauche
171     NEXT
172     MOVER 80,-10               '"redescend a la ligne"
173 NEXT
174 '
175 ' execute symbol
180 SYMBOL ascii,psb(0),psb(1),psb(2),psb(3),psb(4),psb(5),psb(6),
    psb(7)
190 GOSUB 3500                     'affiche le caractere
200 FOR i=0 TO 7                   'affiche les parametres
205     PRINT #1,psb(i)
206 NEXT
210 GOTO 10                        'demande un autre caractere
211 '
212 '
213 '
214 '
215 '=========== sous - programmes ====================
216 '
217 '
997 '
998 ' allume le curseur clignotant
999 '
1000 cur=xc: GOSUB 2000: xc=cur      'sp en 2000 force le curseur
1010 cur=yc: GOSUB 2000: yc=cur      'dans la matrice
1020 MOVE 10*xc+5,10*yc+5
1030 ancien=TESTR(0,0)                'memorise la couleur du carre
1040 PLOTR 0,0,3:PLOTR 1,1:PLOTR -1,0:PLOTR 1,-1  '4 points curseur
1050 RETURN
1497 '
1498 ' efface le curseur clignotant
1499 '
1500 PLOT 10*xc+5,10*yc+5,ancien
```

```
1510 PLOTR 1,1: PLOTR -1,0: PLOTR 1,-1
1520 RETURN
1997 '
1998 ' force le curseur dans la matrice
1999 '
2000 IF cur<0 THEN cur=0
2010 IF cur>7 THEN cur=7
2020 RETURN
3497 '
3498 ' affichage du caractere en mode graphique
3499 '
3500 TAG: PLOT -5,-5,1        'ecriture sous graphique, encrier 1
3510 MOVE 120,47              'curseur.g haut gauche caractere
3520 ascii=INT(ascii)         'si valeur fournie incorrecte
3530 PRINT CHR$(ascii);       'affiche caractere
3540 TAGOFF: RETURN
3997 '
3998 ' recopie caractere dans matrice
3999 '
4000 CLS #1: MOVE 120,32      'curseur.g bas gauche caractere
4010 FOR ym=0 TO 7            'de bas en haut
4020      FOR xm=0 TO 7       'de gauche a droite
4030           IF TESTR(0,0)=1 THEN cm=2 ELSE cm=0
4035           a=XPOS:b=YPOS 'memorise position c.graphique
4040           GOSUB 4500     'recopie dans matrice xm,ym couleur cm
4050           MOVE a+2,b     'taille pixel = 2 pas
4060      NEXT
4070      MOVER -16,2         'ligne au dessus
4080 NEXT: RETURN
4497 '
4498 ' colorier un carre de la matrice xm,ym avec couleur cm
4499 '
4500 FOR k=2 TO 8 STEP 2
4510      PLOT 10*xm+2,10*ym+k,cm: DRAWR 6,0
4520 NEXT: RETURN
4996 '
4997 '
4998 ' initialisation  *************
4999 '
5000 MODE 1:INK 0,1:INK 1,24:INK 2,6:INK 3,26,0 'mode et couleurs
5010 WINDOW #0,1,20,1,25                 'fenetre 0 a gauche ecran
5020 WINDOW #1,26,40,1,9                 'fenetre 1 a droite haut ecran
5030 ORIGIN 400,160,400,540,280,160 'fenetre graphique
5040 PAPER #0,0:PEN #0,1:PAPER #1,0:PEN #1,2    'couleurs texte
5050 DIM psb(7)                          'table parametre SYMBOL
5060 FOR i=0 TO 8                        'trace de la matrice
5070      PLOT 10*i,0,1:DRAWR 0,80       'verticales
5080      PLOT 0,10*i  :DRAWR 80,0       'horizontales
5090 NEXT
5098 '
5099 'reservation memoire caractere uniquement si 1er lancement
5100 INPUT "1er lancement";a$
5110 IF a$<>"o" AND a$<>"n" THEN GOTO 5100
5120 IF a$="o" THEN SYMBOL AFTER 0
5130 RETURN
7998 '
7999 '
```

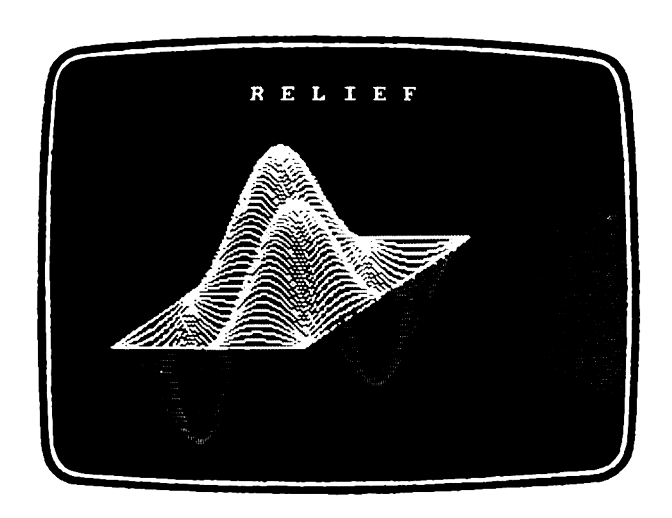
Relief

Le dôme de la cathédrale de Florence, le toit des pagodes chinoises ou celui de votre future maison, voici ce que ce programme se propose de dessiner, si vous connaissez les équations de ces surfaces. Vous pouvez bien entendu l'utiliser si vous avez suffisamment d'intuition et de sens artistique.

Une fois chargé, le programme vous demande d'écrire l'équation de la courbe.

Si vous répondez :	- sin(2*x) * sin(2*y)		
	hauteur :	8Ø	(donne le relief à la courbe)
	largeur :	3ØØ	(définit la perspective)
	pas de x :	2	(définition horizontale)
	pas de y :	1	(définition en "profondeur")

vous obtenez la courbe suivante :



```
1 '                   ***************
2 '                   * R E L I E F *
3 '                   ***************
4 '
5 GOTO 700     ' depart du programme
8  '---------------------------------------------------------------------
9  ' calcul des coordonnees graphiques
10 ' ---------------------------------
20 FOR j = h TO ly STEP py       ' boucle sur l'ordonnee
30   y = ay + (j-h)*dy            '  y de ay a by
40   i0 = (j-h)*lr/ly             '  origine graphique de x
60   FOR i = 0 TO id STEP px      '  boucle sur l'abscisse
70     i1 = i + i0 + xorig        '    abscisse graphique
80     x = ax + i*dx              '    abscisse reelle
85 '
90     z = FN f(x,y)              '    fonction
95 '
100    k = j + h*z                '    ordonnee graphique
110    GOSUB 200                  '    affichage ( i1 , k )
120    ak = k                     '    ancien k
130  NEXT i
140 NEXT j
150 '
160 RETURN
188 '
189 '---------------------------------------------------------------------
190 '    affichage ( i1 , k )
191 '    ---------
200 IF k >= mx(i1)  OR  i=lx GOTO 225     ' point invisible
205  IF k < mn(i1) THEN PLOT i1,k,3       ' autre cote de la courbe
210  GOTO 240                             ' couleur 3
220 '
225 PLOT i1,k,2                           ' point visible : couleur 2
230 '
235 '
240 IF i>lx THEN k=ak-px*ly/lr
245 '
250 FOR t = 1 TO px           ' mise a jour de mx(t) et mn(t)
260  t1 = i1 - px + t         ' entre 2 points
270  IF t1 >= 0 THEN mx(t1) = MAX(ak+(k-ak)*t/px,mx(t1)) :
                     mn(t1) = MIN(ak+(k-ak)*t/px,mn(t1))
280 NEXT t
285 '
290 RETURN
489 '---------------------------------------------------------------------
490 ' initialisation
491 ' --------------
500 INK 0,16 :INK 1,6         ' 0:rose  1:rouge
505 INK 2,2 :INK 3,24         ' 2:bleu  3:jaune
510 WINDOW #1,1,40,1,2
520 '
530 MODE 1 :PAPER #1,0        ' fond rose
535 BORDER 0 :PEN #1,1        ' bord noir,encre rouge
540 '
550 CLS
560 DIM mx(640),mn(640)               ' maxima et minima
570 '
580 ax=0 : bx=PI                      ' extremites axe des x
590 ay=0 : by=PI                      ' extremites axe des y
600 haut=350                          ' hauteur
602 larg=550 :xorig=(640-larg)/2      ' largeur de l'ecran
605 FOR i=1 TO 640: mn(i)=400   :NEXT i
610 '
620 INPUT #1,"Hauteur  (1 a 174)",h :IF h <1 OR h >174 GOTO 620
630 INPUT #1,"Largeur  (1 a 549)",lx:IF lx<1 OR lx>549 GOTO 630
```

```
640 INPUT #1,"Pas de x (1 a 100)",px:IF px<1 OR px>100 GOTO 640
650 INPUT #1,"Pas de y (1 a  50)",py:IF py<1 OR py> 50 GOTO 650
655 CLS #1: PRINT #1:PRINT#1,"              R E L I E F"
660 lr = larg - lx                 ' largeur restante
662 ly = haut - h                  ' hauteur restante
665 '
670 dx =(bx-ax)/lx                 ' pas reel de x
672 dy =(by-ay)/(ly-h)             ' pas reel de y
675 '
680 id = lx + py*lr/ly             '
685 '
686 RETURN
689 '-------------------------------------------------------------
690 '  Depart du programme
691 '  -------------------
700 KEY 138,CHR$(13)+"750 def fn f(x,y) = "    ' touche '.' redefinie
705 KEY 139,CHR$(13)+"run 740"+CHR$(13)        ' touche enter redefinie
710 CLS :PRINT:PRINT:PRINT"            R E L I E F " : PRINT
720 PRINT "      appuyez sur '.'(de droite)"
722 PRINT "puis definissez la courbe "
724 PRINT "et   appuyez sur enter(petit)"
730 END
735 '  -------Nouveau depart ----------
740 ON ERROR GOTO 1000
750 DEF FN f(x,y) =-SIN(2*x)^2*SIN(y)
760 GOSUB 500                          ' initialisation
770 '
780 GOSUB 10                           ' calcul et affichage
785 '
790 KEY 138,"." : KEY 139,CHR$(13)
795 '
800 WHILE INKEY$="" :WEND :     END
810 '
820 '
990 '-----------------------------------------------
991 '   cas d'erreurs
992 '   -------------
1000 CLS : CLS #1
1020 '
1030 '
1050  IF ERR <> 6 AND ERR <> 11 GOTO 1100
1060    PRINT #1,"la fonction genere une erreur de calcul dans
   l'intervalle"
1070    PRINT #1,"  x : ";ax;" - ";bx
1080    PRINT #1,"  y : ";ay;" - ";by
1090    END
1100  PRINT #1,"il y a une erreur ";ERR;" a la ligne ";ERL
1110  END
```

Commentaires

Les lignes 7ØØ à 73Ø redéfinissent les touches [.] et [enter] du pavé numérique. L'utilisateur du programme réécrit en fait la ligne 75Ø et lance le programme à la ligne 74Ø.

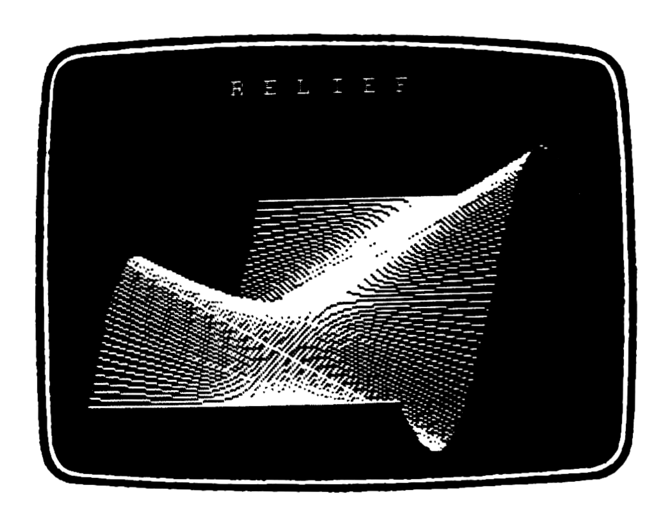
Les matrices mx (64Ø) et mn (64Ø) (ligne 56Ø) représentent les maxima et les minima utiles au calcul des parties cachées et à celui de l'envers de la surface.

Le sous-programme 1Ø-16Ø est le cœur du programme. Il utilise le sous-programme 19Ø-28Ø pour dessiner, ou non, le point qu'il a calculé.

Variations

Les limites sur les axes x et y sont fixées arbitrairement aux lignes 58Ø et 59Ø. Il est possible de modifier ces lignes pour demander à l'utilisateur de choisir lui-même le champ de définition de la surface.

Pour remplir pleinement l'écran sans risque de débordement, il faudrait faire un premier calcul qui détermine le point le plus haut de la courbe. La fonction f(x,y) sera alors multipliée par un facteur qui ajuste cette courbe aux limites de l'écran.



Il peut être intéressant, une fois le dessin achevé, de garder sur cassette ou sur disquette la mémoire écran (adresse &CØØØ à &FFFF) pour ne pas avoir à recalculer tous les points si vous voulez réaffi-

cher une courbe ou si votre talent risque d'inquiéter Vasarely. Il suffit pour cela d'ajouter la ligne suivante à votre programme :

785 SAVE "! image", B, &CØØØ, &3FFF

Grâce au point d'exclamation, aucun texte ne sera écrit sur votre écran qui ne contiendra que la courbe tracée. A cause du même point d'exclamation, aucun dialogue ne sera établi avant l'enregistrement. Il vous faudra donc mettre le magnétophone sur la position "enregistrement" avant de lancer le programme. L'image est restituée par la commande suivante :

LOAD "! image"

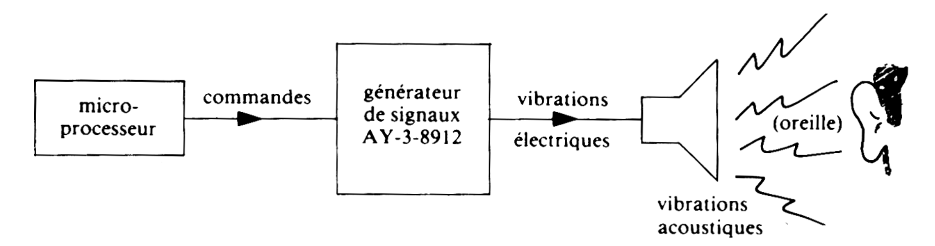
Chapitre II

Plein les oreilles !

Introduction

Un son est une vibration de l'air ressentie, avec plus ou moins de plaisir, par l'oreille humaine ou par une oreille animale.

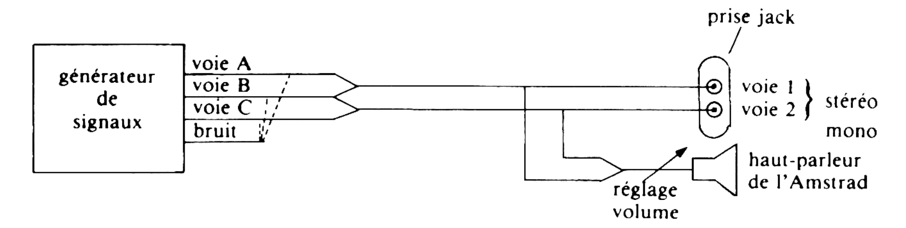
La membrane d'un haut-parleur produit les vibrations acoustiques nécessaires si on lui soumet des vibrations électriques. C'est au générateur de signaux AY-3-8912 que revient la tâche de fournir les vibrations électriques à partir des commandes données par le microprocesseur.



Du microprocesseur à l'oreille.

L'Amstrad produit en fait bien plus de services que le schéma ci-dessus ne le laisse prévoir. En effet, il peut commander trois voies (ou canaux) indépendantes, préciser leur fréquence, volume, durée, variation de fréquence et de volume dans le temps. Il peut aussi mixer à chaque canal un son aléatoire (bruit). Il peut enfin synchroniser tous les canaux pour éviter une certaine cacophonie.

Et tout cela est fait sans perte excessive de temps, grâce aux systèmes de mémorisation des notes et d'interruption sur fin d'exécution.



Les trois voies du générateur.

Les possibilités énumérées peuvent sembler superflues mais nous sommes très loin du gadget. En effet, pour le prix d'un micro-ordinateur, l'Amstrad peut devenir un "synthétiseur" si ce que l'on attend de ses services est précisé avec soin.

La commande SOUND

Cette commande est la principale directive de génération d'un son. Sa syntaxe est la suivante :

SOUND ind.canaux, période, durée, volume, env.vol, env.ton, bruit

Les deux premiers paramètres, ind. canaux et période, sont les seuls obligatoires.

Nous allons détailler chacun des sept paramètres de la commande car leur compréhension permet une utilisation simple et efficace du générateur de son.

"Ind.canaux" est l'indication sur les canaux. Sa valeur est la somme des valeurs qui suivent selon l'effet recherché (il s'agit en fait de positionner les huit bits d'un octet).

1 : Son sur le canal A
2 : Son sur le canal B
4 : Son sur le canal C
8 : Le son attend que le canal A soit activé pour démarrer
16 : Le son attend que le canal B soit activé pour démarrer
32 : Le son attend que le canal C soit activé pour démarrer
64 : Le son attend une commande RELEASE sur les canaux cités (1, 2, 4) pour démarrer
128 : Le son détruit les sons en attente sur les canaux cités et est exécuté immédiatement sauf s'il y a des rendez-vous (8, 16, 32) ou une attente

Tout ceci paraît bien compliqué à première vue, mais il n'en est rien. Quelques exemples vont nous permettre de comprendre ce qui se passe.

Exemple 1

```
SOUND 1,200
```

Le son de période "2ØØ" est exécuté sur le canal A.

Exemple 2

```
SOUND 5,200          ' 1 + 4 = 5
```

Le son de période "2ØØ" est exécuté sur les canaux A et C.

Exemple 3

```
SOUND 17,200         ' 1 + 16 = 17
SOUND 10,300         ' 2 +  8 = 10
```

La première commande dit au canal A (1) d'attendre le canal B (16) pour jouer la note "2ØØ".

La deuxième commande ordonne au canal B (2) d'exécuter la note "3ØØ" et de libérer en même temps le son "2ØØ" qui patientait sur le canal A.

Notons que si A a rendez-vous avec B, B doit avoir rendez-vous avec A pour que les deux canaux puissent jouer.

Exemple 4

```
SOUND 69,200      ' 1 + 4 + 64 = 69
```

Les sons sur les canaux A et C (1 + 4) attendent un ordre RELEASE (64) pour démarrer ensemble :

RELEASE (5) ou RELEASE (1)
puis RELEASE (4)

Exemple 5

```
SOUND 17,200      ' 1 + 16 = 17
```

Cette commande ne génère aucun son car le canal A a rendez-vous avec le canal B (cf. exemple 3).

Si nous donnons maintenant l'ordre

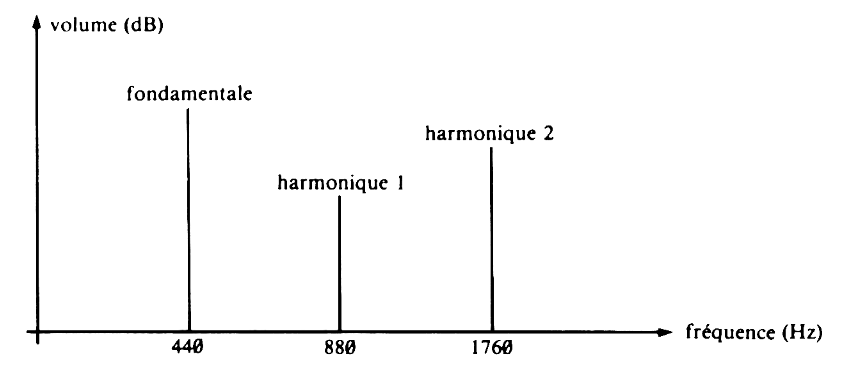
```
SOUND 129,300     ' 1 + 128 = 129
```

la commande précédente de période "2ØØ" est ignorée et le son de période "3ØØ" est exécuté sur le canal A.

Exemple 6

Une utilisation des canaux : le timbre d'un instrument de musique.

Un son possède rarement une fréquence unique (la fondamentale). Les vibrations de fréquence multiple de la fondamentale, appelées les harmoniques, constituent le timbre du son.



Une fréquence fondamentale et deux de ses harmoniques.

En générant la fondamentale sur le canal A et les harmoniques 1 et 2 sur les canaux B et C, nous pouvons reproduire des sons plus ou moins riches en harmoniques et nous rapprocher ainsi de tel ou tel instrument de musique.

Voici un programme qui génère un son avec des harmoniques :

```
10 SOUND 33,284    ' 1+32=33     : le canal A attend le canal C
20 SOUND 34,284/2  ' 2+32=34     : le canal B attend le canal C
30 SOUND 28,284/4  ' 4+8+16=28 : le canal C libere les autres
                canaux et les trois voies jouent au meme moment.
```

"Période"
Cette valeur détermine la hauteur de la note. Elle est reliée à la fréquence par la formule:

$$\text{"période"} = \frac{125\emptyset\emptyset\emptyset}{\text{fréquence}}$$

Voici un programme qui émet les sons des gammes chromatiques à partir du do ("période" = 3822) le plus grave :

```
10 div = 2 ^ (1/12)
20 periode = 3822
30 FOR i = 0 TO 100
40   note = i MOD 12
50   IF note=1 OR note=3 OR note=6 OR note=8 OR note=10 GOTO 70
60   SOUND CINT(periode),50
70   periode = periode / div
80 NEXT i
90 END
```

Il suffit d'éliminer la ligne 5∅ pour entendre tous les demi-tons (do dièse, ré dièse, fa dièse, sol dièse, la dièse).

"Durée"
Ce paramètre fixe la durée de la note avec une unité de dix millisecondes (un centième de seconde). S'il est absent de la commande SOUND, la note est exécutée pendant ∅,2 seconde.

Si "durée" vaut ∅, la note est jouée pendant le temps de l'enveloppe de volume (voir plus loin). Une valeur négative de "durée" représente le nombre de fois où l'enveloppe est répétée, identique à elle-même.

Exemple 1

```
SOUND 1,200,100
```

Son de 1 seconde (100 × 10 ms)

Exemple 2

```
ENV   13 ,1,15,200       ' cette enveloppe dure
SOUND 1,200,0,0,13       ' son de 2 secondes fixe par
                           l'enveloppe 13
SOUND 1,200,-3,0,13      ' son de 6 (3*2) secondes
```

"Volume"
Le paramètre volume donne une valeur de volume dans l'échelle Ø..15. C'est cette valeur qui détermine la valeur initiale de l'enveloppe de volume, si elle existe.

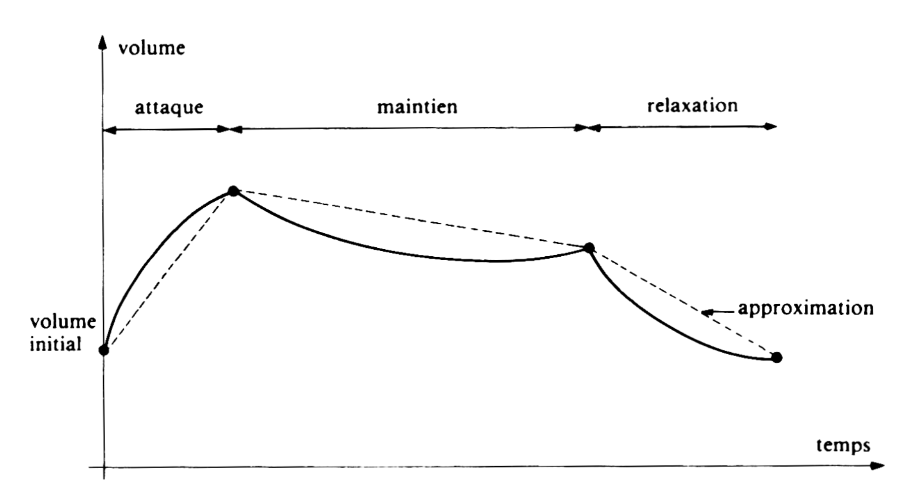
"Env.vol"
"Env.vol" est un numéro d'enveloppe de volume compris entre Ø et 15. La valeur Ø, ou l'absence de valeur, correspond à une enveloppe prédéfinie prise par défaut (volume constant). La commande SOUND dispose donc en permanence des enveloppes auparavant décrites par les commandes ENV (voir plus loin).

"Env.ton"
"Env.ton" est un numéro d'enveloppe de "période" compris entre Ø et 15. La valeur Ø, ou l'absence de valeur, correspond à une enveloppe sans variation de fréquence.

"Bruit"
Ce paramètre, compris entre Ø et 31, permet de mixer une fréquence aléatoire au(x) canal(canaux) cité(s) dans le premier paramètre de SOUND.

L'enveloppe de volume

Cette notion correspond à une approche plus fine de la durée et du volume. En effet, le volume d'une note varie pendant l'exécution de cette note. Il y a en général trois phases qui sont représentées sur la courbe ci-dessous.



Forme type d'une enveloppe.

Une attaque franche convient bien pour un instrument à cordes alors qu'un volume à peu près constant simule correctement un instrument à vent. Cependant, toute courbe étant autorisée, il est possible de créer des sons originaux.

L'Amstrad permet de dessiner approximativement cette enveloppe de volume grâce à cinq droites au maximum. La directive BASIC est :

ENV numenv ,P1,V1,T1 ,P2,V2,T2 ,...

où :

numenv est un numéro d'enveloppe compris entre 1 et 15. Cela permet de constituer une bibliothèque d'enveloppes que la commande SOUND peut utiliser en modifiant uniquement son cinquième paramètre, env.vol.

Pi est le nombre de pas à l'intérieur du i^{ème} segment de droite (section). Pi est compris entre Ø et 127. Une valeur nulle maintient le son au niveau constant Vi pendant le temps Ti. Cette valeur nulle est donc à déconseiller car elle crée des discontinuités de volume.

Vi est la variation unitaire de volume comprise entre − 128 et + 127. Il faut remarquer que le volume varie en réalité entre les niveaux Ø et 15. Les valeurs de Vi donnant pour

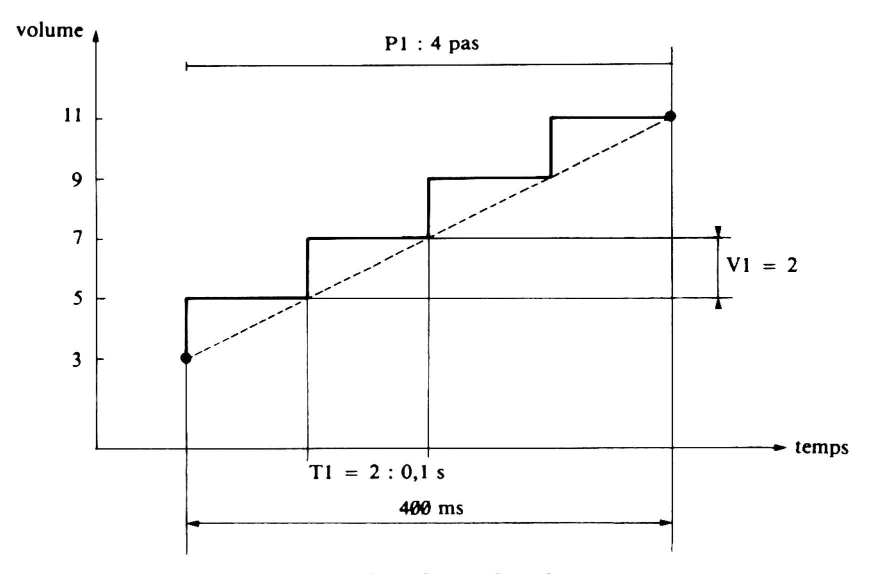
résultat un volume total hors de cette zone (Ø à 15) produisent en fait un volume ramené à cet intervalle par le reste de la division du volume par 16. Ces valeurs "hors zone" créent de brusques variations de volume qui intéresseront les amateurs de sons étranges.

Exemple
Si le volume initial est à 6 et si V1 vaut 12, le volume calculé est 18 et le volume réel est 2 (reste de la division de 18 par 16).

Ti — est le temps de chaque pas à l'intérieur d'une section. Ce temps est compris entre Ø et 255, avec une unité de 1Ø millisecondes. La valeur Ø indique en fait la valeur maximale (2,56 secondes).

Un exemple d'enveloppe de volume :

```
10 ENV  12 ,4,2,10
20 SOUND 1,284,50,3,12
```



Exemple d'enveloppe de volume.

Le volume passe d'un niveau 5 à un niveau 11 pendant Ø,4 seconde, en quatre étapes.

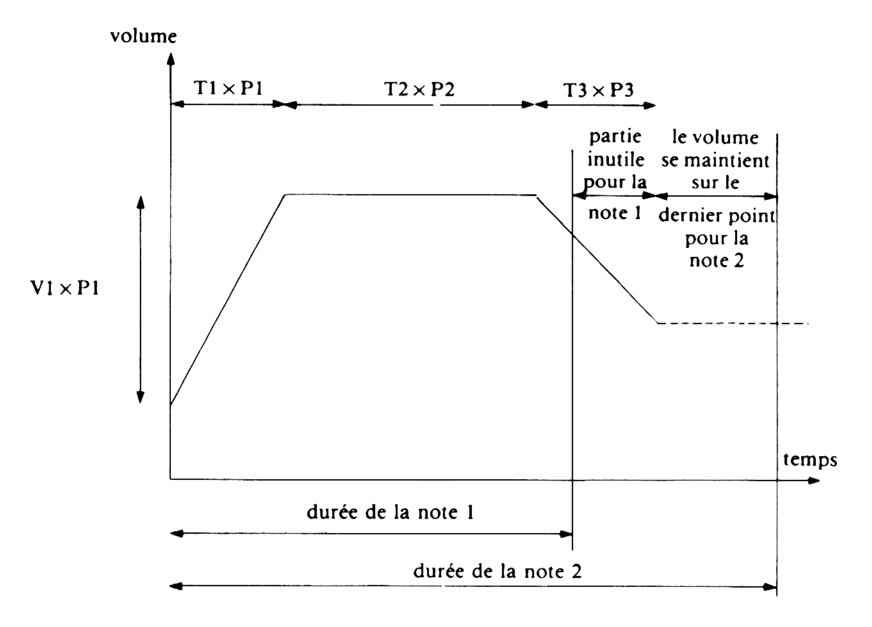
Le volume initial d'une enveloppe est celui donné par le quatrième paramètre de la commande SOUND, augmenté de la valeur V1. Il est conseillé de le fixer à Ø pour ne pas dépasser un volume de 15, à moins que l'on recherche des effets spéciaux.

Un dernier point à préciser est que le temps total de l'enveloppe

T1*P1 + T2*P2 + ...

doit être inférieur à la durée de la note (troisième paramètre de SOUND).

Dans le cas contraire, une partie de l'enveloppe ne servirait à rien.



Enveloppe commune à deux notes de durée différente.

Il existe, pour les connaisseurs, une possibilité de commander directement le générateur de sons AY-3-8912 par l'intermédiaire de ses registres R11 à R13 :

ENV numenv, =R13,R11 – R12 ,...

Exemple

```
ENV 3 , =12,11
SOUND 1,100,200,15,3
```

L'enveloppe de période (de tonalité)

Lorsque le contrebassiste fait trembler le doigt qu'il appuie sur la corde, la note produite voit sa fréquence varier au même rythme. Il s'agit d'un phénomène de vibrato que l'Amstrad peut reproduire par la commande d'enveloppe de tonalité :

ENT nument, P1,G1,T1 ,P2,G2,T2 ,...

Les paramètres Pi et Ti sont définis de la même façon pour les commandes ENV et ENT. Pi peut prendre les valeurs Ø à 239; Ti les valeurs Ø à 255. La valeur Ø pour Ti correspond à 256.

Les valeurs Gi représentent les variations de la "période de tonalité". Ces valeurs sont comprises entre – 128 et 127.

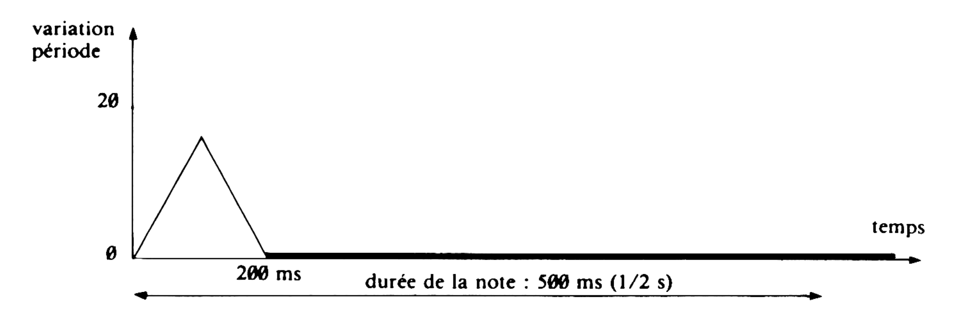
Le numéro d'enveloppe "nument" est compris entre 1 et 15 ou entre – 15 et – 1. Si ce numéro est négatif, l'enveloppe se répète, identique à elle-même, pendant l'exécution du son.

Les deux exemples qui suivent vont nous aider à comprendre la manière de générer des enveloppes de période.

Exemple 1

```
ENT  11 , 5,2,2 , 5,-2,2
SOUND 1,300,50,,,11
```

La commande ENT définit l'enveloppe suivante :



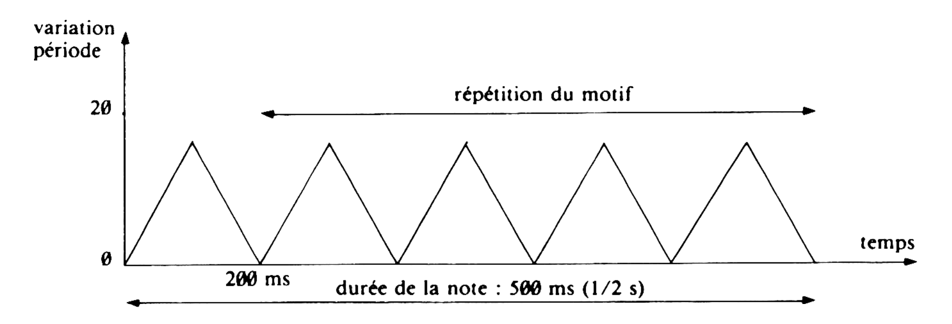
Une enveloppe définie par ENT.

La fréquence varie uniquement au début de la note, ce qui ne constitue pas un réel phénomène de vibrato.

Exemple 2

```
ENT  -11 , 5,2,2 , 5,-2,2
SOUND 1,300,50,,,11
```

La seule différence, mais de taille, avec l'exemple précédent est le signe moins sur le numéro de l'enveloppe (– 11). Ce signe indique que, lors de l'exécution de la note, le motif est répété. Le phénomène de vibrato est donc continu tout au long de la note.



Répétition d'un motif d'enveloppe.

Il existe une syntaxe différente pour les enveloppes de période dont les valeurs réelles de périodes sont données et non leurs variations :

ENT nument, =V1,T1 ,...

Exemple 2

```
ENT 3 , =1000,10
SOUND 1,100,200,15,,3
```

La période "100" n'est pas lue. Seule la valeur 1000 est utilisée.

Remarques générales sur les enveloppes

— Il est possible de détruire les paramètres d'enveloppe de volume ou de période, pour revenir à leurs valeurs par défaut (ni variation de volume, ni variation de période). Il suffit pour cela d'écrire :

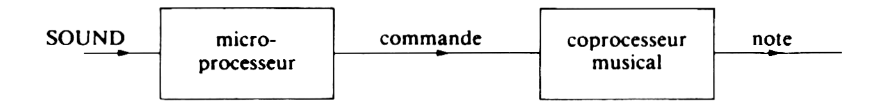
ENV numenv
ou ENT nument

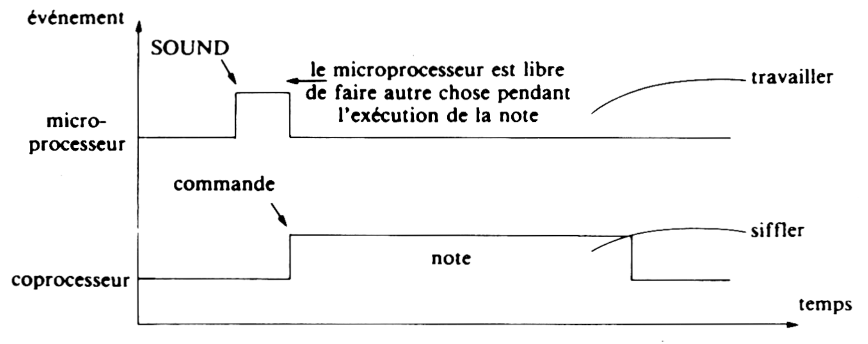
— Si le temps vous manque pour réfléchir à ces problèmes d'enveloppe, vous pourrez générer automatiquement une bibliothèque d'enveloppes grâce aux programmes proposés à la fin de ce chapitre.

Siffler en travaillant : SQ ( )

Ce titre signifie que l'Amstrad, lorsqu'il fait de la musique, peut poursuivre un tout autre travail.

Cela est dû au fait que le microprocesseur est aidé dans sa tâche musicale par le coprocesseur spécialisé AY-3-8912.





Relation entre coprocesseur et microprocesseur.

Par exemple, si vous demandez un son de vingt secondes par

```
SOUND 1,284,2000
```

le message "Ready" apparaît tout de suite et vous continuez à entendre le *la* international même si vous faites un "list" ou toute autre commande (excepté un SOUND 129,Ø).

Le microprocesseur n'attend donc pas la fin de la note pour exécuter une nouvelle instruction, et continue à travailler pendant les vingt secondes durant lesquelles la note est jouée.

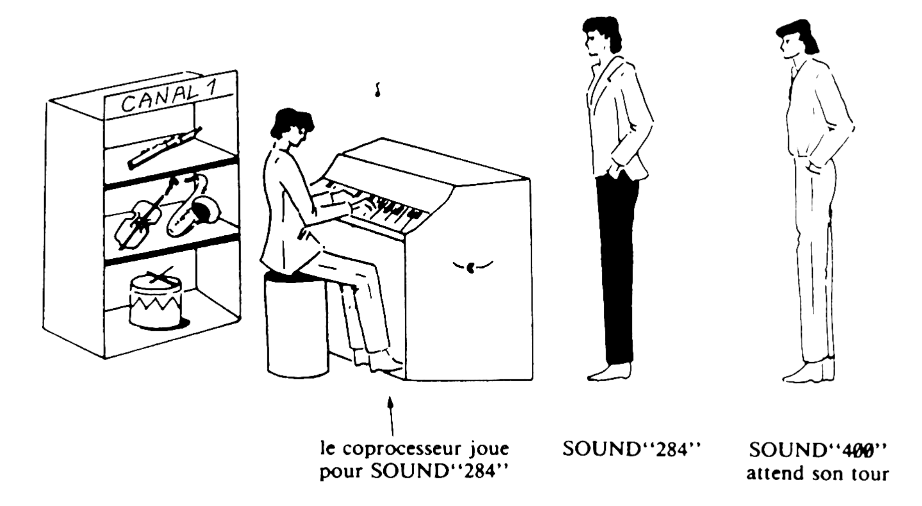
Mais l'Amstrad ne s'arrête pas là pour autant.

Que se passe-t-il lorsque deux sons doivent être joués sur le même canal, l'un à la suite de l'autre?

```
10 SOUND 1,284,2000
20 SOUND 1,400,2000
```

A la lecture de la première commande, SOUND "284", le microprocesseur demande à son coprocesseur musical de jouer la note "284", et ce dernier s'exécute immédiatement.

Le BASIC trouve ensuite à la ligne 2Ø une nouvelle commande pour le même canal. Or ce canal est occupé à jouer la note "284". Le microprocesseur, sachant cela, va mettre en file d'attente la deuxième note ("4ØØ").

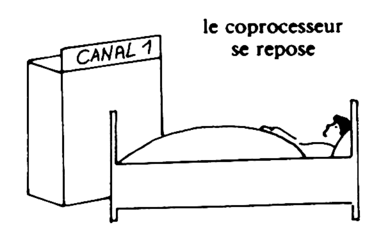


File d'attente devant le coprocesseur.

Pendant ce temps, le microprocesseur est libre de vaquer à diverses occupations.

Lorsque SOUND "284" a fini de faire travailler le canal 1, le coprocesseur regarde s'il y a un autre client, un autre son à émettre, sur le même canal. Il s'aperçoit que SOUND "4ØØ" attend son tour. C'est donc lui qui va faire jouer le coprocesseur pendant que le microprocesseur est toujours libre de ses gestes.

Une fois SOUND "4ØØ" satisfait, le coprocesseur regarde à nouveau s'il lui reste un client à servir. Or il n'y a plus de SOUND en attente. Le coprocesseur peut donc dormir jusqu'au prochain SOUND.





« Partage du travail ! »

Ce système est valable pour les trois canaux. Sur chaque canal, il peut y avoir au maximum un son en cours d'exécution et quatre sons en attente. Nous rappelons que les trois canaux ont pour numéro 1, 2 et 4 et non 1, 2 et 3.

CANAL 1

SOUND
exécuté

SOUND
en attente

CANAL 2

CANAL 4

Files d'attente devant les canaux.

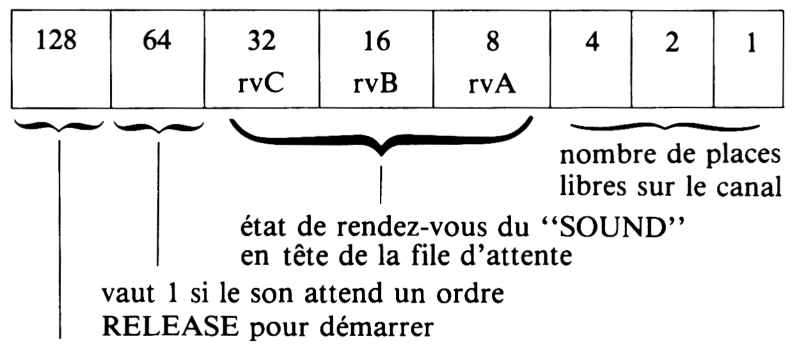
Le programme suivant met en évidence le système de file d'attente :

```
10 d=200              ' duree : 2 secondes
20   FOR i=1 TO 5     ' cinq sons pour le canal 1
30     SOUND "1,10*i,d
40     PRINT i
50   NEXT i
60 PRINT "fin"
70 END
```

En augmentant, à la ligne 2Ø, la valeur extrême de la boucle, nous vérifions que le microprocesseur ne peut mettre en attente plus de quatre sons sur un canal.

Il existe trois variables que l'on peut interroger pour connaître l'état des trois canaux. Ce sont SQ(1), SQ(2), SQ(4). S et Q sont les initiales de SOUND QUEUE, file d'attente de sons.

Les trois variables SQ sont des octets ainsi disposés :



vaut 1 si le son est en cours d'exécution

Exemple 1

```
10 SOUND 1,284
20 PRINT SQ(1),SQ(2),SQ(4)
RUN
132       4       4
```

La valeur 128 (132 = 128 + 4) sur le canal 1 montre que celui-ci exécute une note au moment du "PRINT".
Les valeurs 4 (132 = 128 + 4) restantes indiquent qu'il reste quatre places libres sur chaque canal.

Exemple 2

```
10 SOUND 17,284        ' 17=1+16 : le canal A a rendez-vous avec
                                   le canal B
20 SOUND 10,300        ' 10=2+8  : le canal B a rendez-vous avec
                                   le canal A
30 PRINT SQ(1),SQ(2),SQ(4)
RUN
132       132       4
```

Les canaux A et B sont occupés et il reste quatre places libres sur chaque canal.

Exemple 3

```
10 SOUND 1,284
20 PRINT SQ(1),SQ(2),SQ(4)
30 SOUND 1,400
40 PRINT SQ(1),SQ(2),SQ(4)
RUN
132        4         4
131        4         4
```

Lors du "PRINT" de la ligne 2Ø, il reste quatre places libres sur chaque canal (132 – 128 = 4). Le "SOUND" de la ligne 3Ø occupe une place dans la file d'attente et il ne reste plus que trois places sur le canal A lors du "PRINT" de la ligne 4Ø.

Exemple 4

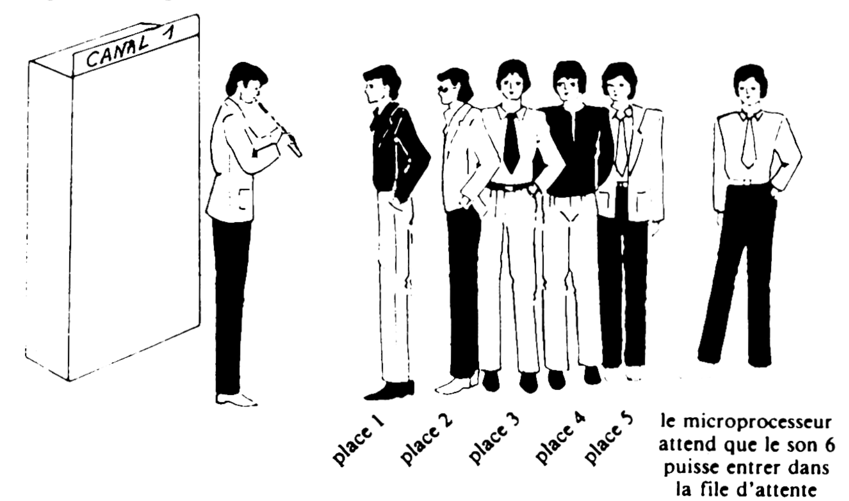
```
10 SOUND 65,284          ' 65 = 64 + 1
20 PRINT SQ(1)
30 RELEASE(1)
40 PRINT SQ(1)
RUN
67
132
```

Le son, généré par la ligne 1Ø, attend une commande de déblocage (64). Il ne reste donc plus que trois places libres (64 + 3 = 67).

La ligne 3Ø libère le canal A. Ce canal peut donc jouer (128). Les quatre places de la file d'attente sont alors libres (128 + 4 = 132).

Prévenez-moi s'il reste une place : ON SQ( ) GOSUB...

Le système des files d'attente est bien pratique lorsque le nombre de notes à jouer n'est pas très élevé. *Au-delà, le microprocesseur est bloqué et ne peut exécuter autre chose.*



Six notes en attente devant le canal 1.

Heureusement, il existe un système qui permet de prévenir le microprocesseur chaque fois qu'il reste une place libre dans la file. La commande BASIC qui active ce système est :

ON SQ(i) GOSUB ligne   i = 1, 2 ou 4

Il suffit donc que le sous-programme appelé mette le son "6" en attente dans la file et réactive le mécanisme par un nouvel ordre "ON SQ(i) GOSUB".

Il existe bien entendu l'équivalent pour les canaux 2 et 3 (ON SQ(2), ON SQ(4)).

Il est important de noter que les commandes SOUND, ainsi que le passage dans le sous-programme appelé, désarment le mécanisme d'interruption en cas de place libre dans la file d'attente. Il est donc obligatoire, s'il reste des notes à jouer, de mettre une commande ON SQ( ) à la fin du sous-programme ou dans le programme principal.

Exemple 1

```
10 GOSUB 100
20 WHILE jamais = 0         ' boucle sans fin
30   PRINT "J'ecris et je joue de la musique en meme temps"
40 WEND
60 '
100 SOUND 1,50+INT(RND(6)*1000)    ' sous-programme musical
110 ON SQ(1) GOSUB 100
120 RETURN
```

L'appel du sous-programme 100 à la ligne 10 active le premier son aléatoire. Le sous-programme est relancé (ON SQ(1) GOSUB 1000) dès que la file d'attente du canal A n'est pas pleine, c'est-à-dire tout de suite. Ce mécanisme est répété pour les cinq premières notes. Il y a ensuite retour à la ligne 30 qui affiche, indéfiniment, son message. Cette boucle est interrompue de temps en temps, lorsqu'une place se libère dans la file d'attente.

Exemple 2

```
10 MODE 2 : CLS
20 ' execution "simultanee" d'un programme et d'une gamme
30 ' ------------------------------------------------------
40 '
50 '
60 MODE 2 : CLS
70 '
80  GOSUB 250              ' lancement de la gamme
90 '
100 '
110  DIM pascal(10,10) : pascal(0,1)=1
120 '
130  FOR i = 1 TO 10        ' triangle de pascal
140   FOR j = 1 TO i
150    pascal(i,j) = pascal(i-1,j-1) + pascal(i-1,j)
160    PRINT USING "####" ; pascal(i,j) ;
170  NEXT j
180  PRINT
190 NEXT i
200 END
210 '
220 '
230  ' sous-programme musical
240  ' ---------------------
250 READ n,d               ' lecture note et duree
260 '
270 IF n = -1 GOTO 300  ' fin de la melodie
280   SOUND 1,478/2^(n/12),50*d
290   ON SQ(1) GOSUB 250   ' on relance la machine
300 RETURN
```

```
310 '
320 '
330 DATA 1,1,3,1,5,1,6,1,8,1,10,1,12,1,13,1,-1,-1
```

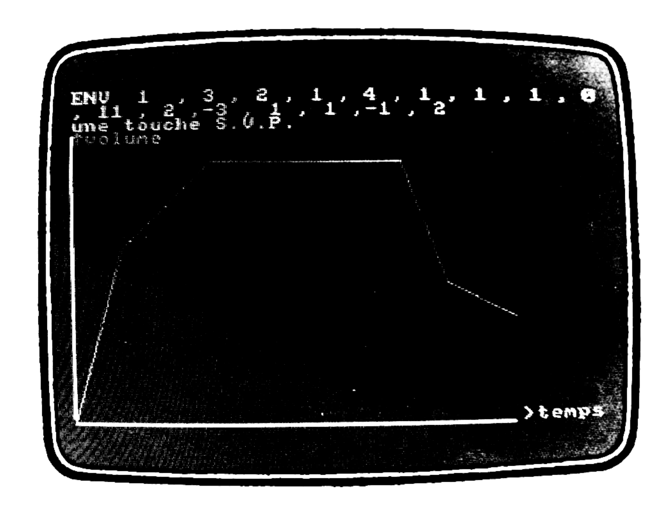
Le principe de ce programme est le même que dans l'exemple précédent. Remarquons simplement qu'il y a une condition d'arrêt à l'appel du sous-programme musical (ligne 27Ø).

Si la boucle de la ligne 13Ø est

FOR i = 1 TO 2

nous n'avons que les cinq premières notes de la gamme car le programme principal ne dure pas suffisamment longtemps pour être interrompu par la libération de la cinquième place de la file d'attente.

Bibliothèques d'enveloppes



La grande liberté de choix des enveloppes de volume et de période se paie par la complexité du calcul des directives ENV et ENT. Nous vous proposons deux programmes qui, à partir de votre dessin, calculent les enveloppes et exécutent un air de musique d'après ces enveloppes. Vous avez ainsi le choix des enveloppes et la simplicité d'utilisation.

Vous choisissez d'abord un numéro d'enveloppe et quelques options dont nous parlons un peu plus loin. Les quatre flèches servent à déplacer votre "crayon" sur l'écran. Une pression sur la touche [copy] valide un point de l'enveloppe. Lorsque tous les points sont positionnés (cinq au maximum), la touche [enter] lance l'exécution d'une mélodie et affiche la valeur des paramètres calculés. Dès que votre bibliothèque d'enveloppes est constituée, vous pouvez la conserver sur cassette et la lire ultérieurement.

Le générateur d'enveloppe de volume vous pose les questions suivantes :

• effets spéciaux (o/n) ?

Une réponse négative maintient le volume dans les limites Ø à 15. Une réponse positive définit une plage de volume pouvant aller jusqu'à 127 d'après la réponse à la question qui suit.

• répétition volume (Ø à 7) ?

Cette valeur détermine le nombre de fois où le volume passera brusquement de la valeur 15 à la valeur Ø si toute l'échelle de volume est utilisée.

• temps maximum en 1/1ØØ s. (1 à 255) ?

Ce temps définit la pleine échelle de temps sur la courbe. Comme les notes de la mélodie ont une durée de l'ordre de 5Ø millisecondes, il est conseillé de choisir une grandeur qui ne s'éloigne pas trop de cette valeur.

Cette dernière question est également posée par le générateur d'enveloppe de période qui vous aura demandé auparavant :

• variation maximum (Ø à 127)?

Plus cette valeur est élevée, plus le phénomène de vibrato est important, jusqu'à rendre méconnaissable la mélodie.

• répétition (o/n)?

Il est possible de définir un motif qui se répète sans avoir à le dessiner tout au long de la note.

```
1        '****************************************
2        '* Bibliotheque d'enveloppes de volume *
3        '****************************************
4 '
10 GOSUB 3000                    ' initialisation
25 '
30 WHILE jamais = 0              ' boucle "sans fin"
40 '
45   CLS #1
50   PRINT #1,"1 a 15 :Numero d'enveloppe "
51   PRINT #1," 16 : lecture ; 17 : enregistrement "
52   INPUT #1," -1 : fin       " ; numenv
53   IF numenv = -1 THEN    END                ' fin
54   IF numenv<1 OR numenv>17 THEN GOTO 50
55   IF numenv = 17 THEN GOSUB 2000 : GOTO 50 ' enregistrement
57   IF numenv = 16 THEN GOSUB 2500 : GOTO 50 ' lecture cassette
60   IF deja(numenv) = 0 THEN deja(numenv) = 1 : GOSUB 100
65 '
70   GOSUB 1000             ' M E L O D I E
75 '
80 WEND
90 '
95 '
98 ' d e f i n i r   e n v e l o p p e
99 ' ********************************
100 n = numenv
110 INPUT #1,"effets speciaux (O/N)";a$
115 '
120 IF LOWER$(a$)="o" THEN INPUT #1,"repetition volume(0 a 7)";rep
                      ELSE rep=0
122 IF rep>7 OR rep<0 THEN GOTO 120
                      ELSE vmax=16*(rep+1)-1   ' volume max
124                                        ' temps pleine echelle
125 INPUT #1,"temps maximum en 1/100 s.(1 a 255)";tmax
127 IF tmax<1 OR tmax>255 GOTO 125
130 '
140 GOSUB 700                  ' dessin abscisse et ordonnee
150 '
160 PRINT #1,"fleches pour deplacement"
162 PRINT #1,"copy    pour valider un point"
164 PRINT #1,"enter   pour ecouter"
170 '
180 npt = 0                    ' nombre de points
185 t = 0 : v = 0              ' abscisse  ordonnee
190 a$ = INKEY$ : IF a$ = "" THEN GOTO 190 ELSE a = ASC(a$)
195 '
200 d = 10 : tmin = 0         ' deplacement curseur : non retour
210 WHILE a <> 13  AND npt < 5
```

```
220   IF a=240 THEN v=v+d : v=MIN(v,330) : GOTO 350     ' haut
230   IF a=241 THEN v=v-d : v=MAX(v,0)   : GOTO 350     ' bas
240   IF a=243 THEN t=t+d : t=MIN(t,639) : GOTO 350     ' droite
250   IF a=242 THEN t=t-d : t=MAX(t,tmin): GOTO 350     ' gauche
260   IF a<> 224 THEN GOTO 350                          ' copy
270     MOVE tt(npt,n),vv(npt,n)
280     tmin = t                  ' pour ne pas revenir en arriere
290     npt = npt + 1             ' nombre de points
300     tt(npt,n) = t : vv(npt,n) = v
310     DRAW t,v,3  :  GOSUB 400  ' dessin et calcul
320 '
330 '
350   PLOT at,av,0 :at = t : av= v : PLOT t,v,2
360   a$ = INKEY$ : IF a$ = "" GOTO 350 ELSE a = ASC(a$)
370 WEND
375 RETURN
380 '
385 '
390 '
398   calcul d'un point
399 ' *****************
400 IF npt <= 0 GOTO 560
410 t0=tt(npt-1,n) : t1=tt(npt,n)
420 v0=vv(npt-1,n) : v1=vv(npt,n)
430 dt=(t1-t0)/639*tmax             ' delta t
440 dv=(v1-v0)/330*vmax             ' delta v
445 dv = MIN (dv,vmax)              ' volume max pleine
450 '                               ' echelle modulo 16
460 nbpas = MIN (dt,ABS(dv),126)
465 nbpas = MAX (1,nbpas)           ' pas <> 0
470 nbpas(npt,n)= INT(nbpas)
480 '
485 IF nbpas<>0 THEN dv= INT(dv/nbpas) : dt=INT(dt/nbpas)
490 dv=MIN(dv,vmax):dv=MAX(dv,-vmax-1)
500 delta(npt,n) = CINT(dv)         ' variation volume
510 '
520 dt=MIN(dt,255)                  ' delta t
525 IF dt=0 THEN dt=1               ' car 0 represente 256
530 pas(npt,n) = CINT(dt)           ' pas (sur t)
540 '
545 npt(n) = npt
550 '
560 RETURN
570 '
698 ' a x e s
699 ' *******
700 CLG        : PEN 3
710  MOVE 0,0 : DRAW 0,330,2 : LOCATE 1,5   : PRINT "^volume"
720  MOVE 0,0 : DRAW 520,0,2 : LOCATE 34,25 : PRINT ">temps"
730 MOVE 0,0 : PEN 1
740 RETURN
980 '
981 '
990  ' r e s u l t a t s
991  '*******************
1000 n = numenv : npt = npt(n)
1002 GOSUB 700    ' axes
1004 '
1005 PRINT #1,"ENV ";numenv;" ";
1010 FOR i=1 TO npt                      ' boucle sur les points
1015   DRAW tt(i,n),vv(i,n),3            ' dessine les droites
1020   PRINT #1,",";nbpas(i,n);",";delta(i,n);",";pas(i,n);
1030 NEXT
1040 PRINT #1,""
1045 '
```

```
1050 ON npt GOTO 1060,1065,1070,1075,1080
1060 ENV numenv,nbpas(1,n),delta(1,n),pas(1,n) : GOTO 1100
1065 ENV numenv,nbpas(1,n),delta(1,n),pas(1,n),nbpas(2,n),
  delta(2,n),pas(2,n)  : GOTO 1100
1070 ENV numenv,nbpas(1,n),delta(1,n),pas(1,n),nbpas(2,n),
  delta(2,n),pas(2,n),nbpas(3,n),delta(3,n),pas(3,n) : GOTO 1100
1075 ENV numenv,nbpas(1,n),delta(1,n),pas(1,n),nbpas(2,n),
  delta(2,n),pas(2,n),nbpas(3,n),delta(3,n),pas(3,n),nbpas(4,n),
  delta(4,n),pas(4,n)  : GOTO 1100
1080 ENV numenv,nbpas(1,n),delta(1,n),pas(1,n),nbpas(2,n),
  delta(2,n),pas(2,n),nbpas(3,n),delta(3,n),pas(3,n),nbpas(4,n),
  delta(4,n),pas(4,n),nbpas(5,n),delta(5,n),pas(5,n)
1085 '
1100 RESTORE : READ p : READ d    ' periode - duree
1110 WHILE p <> -1
1120   SOUND 7,p,50/d,0,numenv    ' musique
1130   READ p : READ d
1140 WEND
1145 PRINT #1,"une touche S.V.P." :WHILE INKEY$="" : WEND
1150 RETURN
1160 '
1170 DATA 319,1 ,284,1 ,253,2 ,213,2 ,213,2 ,190,2
1180 DATA 213,2 ,253,2 ,319,3
1300 DATA -1,-1
1980 '
1981 '
1990 'enregistrement
1991 '=============
2000 CLS : INPUT #1,"Enregistrement (REC-PLAY) puis nom";a$ : CLS
2010 OPENOUT a$
2015 '
2020 FOR j = 1 TO 15
2030   PRINT #9,deja(j): PRINT #9,npt(j)
2040   FOR i = 1 TO 5
2050     PRINT#9,nbpas(i,j):PRINT#9,delta(i,j)
2055     PRINT#9,pas(i,j):PRINT#9,tt(i,j):PRINT#9,vv(i,j)
2060   NEXT
2070 NEXT j
2075 '
2080 CLOSEOUT
2090 RETURN
2480 '
2490 'lecture cassette
2491 '================
2500 CLS : INPUT #1,"Lecture (PLAY) puis nom";a$ : CLS
2510 OPENIN a$
2515 '
2520 FOR j = 1 TO 15
2530   INPUT #9,deja(j): INPUT #9,npt(j)
2540   FOR i = 1 TO 5
2550     INPUT#9,nbpas(i,j):INPUT#9,delta(i,j)
2555     INPUT#9,pas(i,j):INPUT#9,tt(i,j):INPUT#9,vv(i,j)
2560   NEXT i
2570 NEXT j
2575 '
2580 CLOSEIN
2590 RETURN
2987 '
2988 '
2990 ' i n i t i a l i s a t i o n s
2991 ' ---------------------------
3000  DIM deja(15) ,nbpas(5,15) , delta(5,15) , pas (5,15)
3005  DIM tt(5,15) ,vv(5,15) ,npt(15)
3006 '
3010 INK 0,1 : BORDER 1             ' bleu fonce
```

```
3020 INK 1,18                          ' vert vif
3030 INK 2,26                          ' blanc
3040 INK 3,6                           ' rouge
3050 MODE 1 : PRINT CHR$(22) + CHR$(1) ' transparence
3060 '
3070 WINDOW #1,1,40,1,4                ' texte
3080 '
3090 PAPER #1,0 : CLS #1               ' papier bleu fonce
3100 PEN #1,1                          ' stylo vert vif
3110 '
3120 RETURN
```

Le programme d'enveloppe de période (ou tonalité) est identique au précédent, excepté les modifications suivantes :

```
1        '****************************************
2        '*  Bibliotheque d'enveloppes de  ton  *
3        '****************************************
4 '
110 INPUT #1,"variation maximum(0 a 127)";vmax
112 IF ABS(vmax) >127 GOTO 110
115 '
120 INPUT #1,"repetition (o/n)";rep$
122 IF LOWER$(rep$)="o" THEN rep = -1  ELSE rep = +1

175 v=330/2 : t=0 : MOVE t,v' origine au milieu

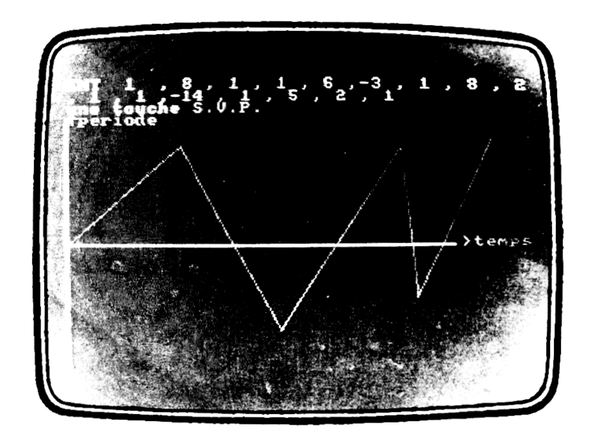
465 '

710  MOVE 0,0   : DRAW 0,330,2 : LOCATE 1,5    : PRINT "^periode"
720  MOVE 0,330/2: DRAWR 520,0,2 : LOCATE 34,15 : PRINT ">temps"
730 MOVE 0,330/2 : PEN 1

1005 PRINT #1,"ENT ";numenv*rep;" ";

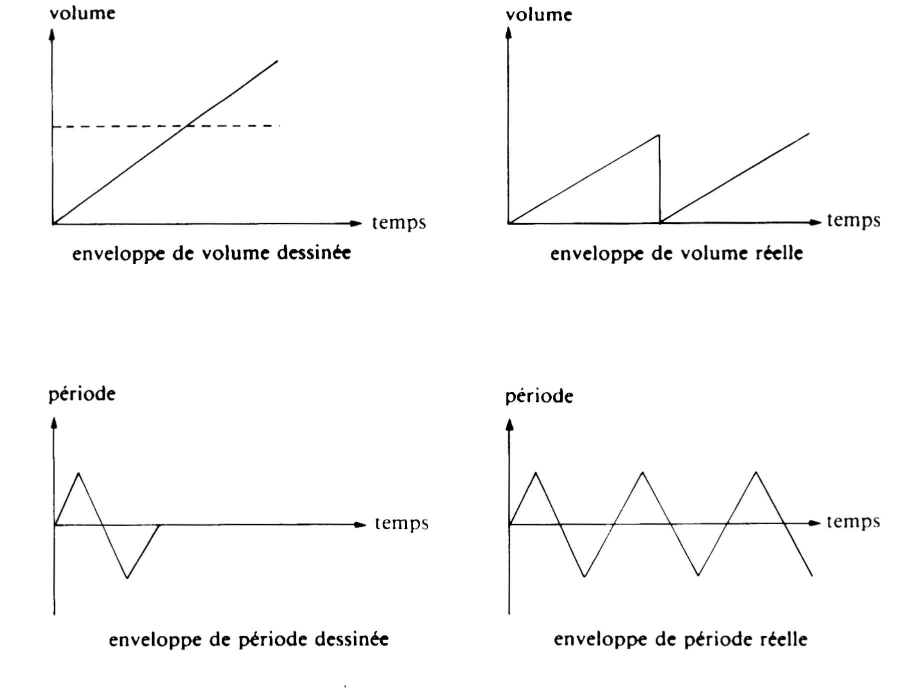
1047  nr = numenv*rep
1050 ON npt GOTO 1060,1065,1070,1075,1080
1060 ENT nr,nbpas(1,n),delta(1,n),pas(1,n) : GOTO 1100
1065 ENT nr,nbpas(1,n),delta(1,n),pas(1,n),nbpas(2,n),
  delta(2,n),pas(2,n)  : GOTO 1100
1070 ENT nr,nbpas(1,n),delta(1,n),pas(1,n),nbpas(2,n),
  delta(2,n),pas(2,n),nbpas(3,n),delta(3,n),pas(3,n) : GOTO 1100
1075 ENT nr,nbpas(1,n),delta(1,n),pas(1,n),nbpas(2,n),
  delta(2,n),pas(2,n),nbpas(3,n),delta(3,n),pas(3,n),nbpas(4,n),
  delta(4,n),pas(4,n)  : GOTO 1100
1080 ENT nr,nbpas(1,n),delta(1,n),pas(1,n),nbpas(2,n),
  delta(2,n),pas(2,n),nbpas(3,n),delta(3,n),pas(3,n),nbpas(4,n),
  delta(4,n),pas(4,n),nbpas(5,n),delta(5,n),pas(5,n)

1120   SOUND 7,p,50/d,15,,numenv ' musique
3007  FOR i=1 TO 15: vv(0,i)=330/2: NEXT  'origine au milieu
```



Variations

— Il peut être intéressant de tracer l'enveloppe réelle et non l'enveloppe dessinée par l'utilisateur (lignes 1Ø1Ø à 1Ø3Ø). Ces deux enveloppes diffèrent uniquement si l'option "répétition" a été choisie.



Passage d'une enveloppe dessinée à une enveloppe réelle.

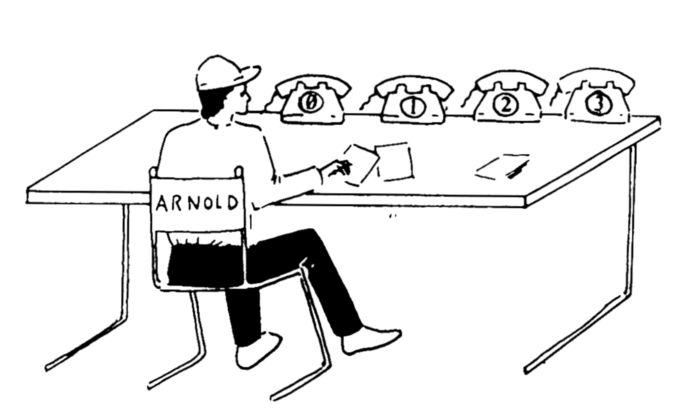
— Il est possible de fusionner ces deux programmes. Le programme résultant donnera la possibilité de faire varier en même temps les enveloppes de volume et de période.
— Vous pouvez développer une option supplémentaire qui consiste à effacer une enveloppe déjà écrite mais qui ne vous satisfait pas. Il suffit pour cela de mettre à zéro la variable "deja" (deja (numenv) = Ø).
— La commande du curseur peut se faire par Joystick en modifiant les lignes 19Ø à 36Ø.

Chapitre III

Les interruptions ou la vie d'Arnold

La vie d'Arnold

Arnold est employé par la société Express pour apposer des tampons sur les documents qui lui sont remis. Il est en liaison directe avec ses quatre chefs hiérarchiques par l'intermédiaire de quatre téléphones disposés sur son bureau. En outre, il est responsable de la sécurité. Une alarme l'avertit en cas de danger.



Arnold à son bureau.

Lorsque la sonnerie de l'un des téléphones retentit, Arnold arrête son travail, prend le combiné et écoute les ordres de l'un de ses chefs. Il effectue le travail demandé et retourne mettre des tampons sur ses fiches. Il arrive que deux téléphones sonnent en même temps. Dans ce cas, Arnold décroche celui qui est relié au plus important des deux chefs. Priorité à la hiérarchie !

Il arrive aussi qu'un grand chef appelle pendant qu'un petit chef demande un travail à Arnold. Dans ce cas, Arnold fait patienter le petit chef, effectue le travail pour le grand chef et reprend la communication avec le petit chef. Si l'inverse arrive, la demande du petit chef est mise sur répondeur téléphonique puis exécutée une fois le travail plus prioritaire effectué.

Ce système fonctionne bien tant que les différents ordres ne sont pas contradictoires. Pour éviter cela, Arnold a la possibilité de placer tous ses téléphones sur répondeur, lorsqu'il sait que le travail qu'il effectue peut être perturbé par un autre appel.

Seul le système d'alarme, plus prioritaire que ses quatre chefs, ne peut laisser Arnold indifférent car, quoi qu'il fasse, il est obligé d'entendre la sirène.

Les interruptions logicielles

Les interruptions gérées par le BASIC de l'Amstrad fonctionnent sur le principe des téléphones d'Arnold. Ce sont des interruptions logicielles, par opposition aux interruptions matérielles (voir chapitre V). Une interruption logicielle suspend le déroulement d'un programme BASIC pour exécuter un sous-programme dit "sous-programme d'interruption".

Il existe trois types d'interruption :
— les réveils, réglés par le programme lui-même,
— les interruptions dues aux canaux sonores,
— le "break" généré par la touche ESC.

Toutes ces interruptions n'ont pas même priorité :

Priorité croissante		
↑	break	Alarme
	réveil 3	P. _ D.G.
	réveil 2 et musique	Chef de division
	réveil 1	Chef de service
	réveil Ø	Chef de groupe
	travail en cours	Arnold

Il est possible de ne pas être dérangé par toutes ces interruptions, excepté le "break". Ces interruptions sont dites masquables. Il se passe alors la même chose que lorsqu'Arnold met ses téléphones sur répondeur. Les interruptions sont mémorisées (enregistrées), puis restituées une fois les masques enlevés. Le nombre maximum d'interruptions enregistrées est de 127.

Nous allons détailler les commandes BASIC qui gèrent les interruptions.

Ces commandes sont de quatre types :
— demande d'interruption,
— suspension d'interruption,
— reprise d'interruption,
— arrêt d'interruption.

Demande d'interruption

Différée

AFTER temps (, réveil) **GOSUB** ligne.
Cette commande demande que le réveil (Ø à 3) appelle le sous-programme dont le numéro de ligne est "ligne" après le temps précisé. Si le numéro de réveil n'est pas donné, le réveil utilisé est celui de plus faible priorité (Ø). L'unité de temps est 2Ø millisecondes (1/5Ø^e de seconde).

Exemple

```
10 AFTER 50 GOSUB 100        ' reveil apres 1 seconde
20 '
30 FOR i = 1 TO 1000         ' travail courant
40   PRINT i
50 NEXT i
60 END
70 '
100 PRINT "Je suis fatigue de compter"
110 PRINT "Appuyer sur ENTER pour que je continue"
120 INPUT a$
130 RETURN
```

Périodique

EVERY temps (, réveil) **GOSUB** ligne.
Si le programme veut être interrompu périodiquement, cette commande lui permet de donner le temps entre deux interruptions, le numéro de réveil utilisé et le sous-programme à exécuter.

Exemple

```
10 EVERY 50 GOSUB 100      ' reveil toutes les secondes
20 '
30 FOR i = 1 TO 1000       ' travail courant
40   PRINT i
50 NEXT i
60 END
70 '
100 PRINT "J'arrete de compter un instant"
110 RETURN
```

Le programme qui affiche les nombres de 1 à 1ØØØ est interrompu toutes les secondes pour écrire "J'arrête de compter un instant".

Remarque

Pour chacun des quatre réveils, il existe un compteur et un seul, indépendant des trois autres. Cependant, les commandes EVERY et AFTER utilisent le même compteur pour un réveil donné. Une commande AFTER ou EVERY sur le réveil i annule donc l'effet d'une commande AFTER ou EVERY sur le même réveil.

Musicale

ON SQ(i) **GOSUB** ligne i = 1, 2 ou 4.
Cette commande est décrite en détail au chapitre II. Elle demande l'exécution du sous-programme "ligne" lorsque le canal i n'est pas plein.

Obligatoire

ON BREAK GOSUB ligne
Le "break" permet, normalement, à l'utilisateur d'un programme d'en arrêter l'exécution à tout instant. La commande ci-dessus contraint le BASIC à exécuter le sous-programme "ligne" en cas de "break".

Exemple

```
10 ON BREAK GOSUB 100
20 '
30 PRINT "Vous ne pourrez plus sortir de ce programme"
40 GOTO 30
50 '
60 '
100 PRINT "Il y a bien eu break"
110 RETURN
```

Attention à ce programme. Il boucle indéfiniment et rend le "break" inopérant.

Suspension d'interruption

DI (Disable Interrupt)
Cette commande rend le BASIC "sourd" aux appels des quatre réveils. Les 127 premiers appels ne sont pourtant pas perdus et sont honorés dès que les interruptions ne sont plus suspendues.

Exemple

```
10 EVERY 50 GOSUB 100
20 '
30 FOR i=1 TO 10000
40   IF a$ = "n" THEN    DI
50   PRINT i
60 NEXT i
70 PRINT "c'est fini"
80 END
90 '
100 INPUT "voulez-vous continuer a etre interrompu";a$
110 RETURN
```

La commande DI de la ligne 4Ø arrête le réveil activé à la ligne 1Ø.

INPUT...
L'appel d'un sous-programme d'interruption (réveil ou musique) ne peut se faire pendant qu'une commande BASIC est en cours d'exécution. La seule commande qui peut bloquer les interruptions est la commande INPUT car son temps d'exécution dépend de l'utilisateur du programme.

En outre, le "BREAK", pendant une commande INPUT, ne peut être traité par interruption et fait arrêter le programme.

Nous vous conseillons de remplacer la commande

```
INPUT a$
```

par la suite de commandes suivantes :

```
a$ = "" : x$ = ""
WHILE x$ <> CHR$(13)
  x$=INKEY$ : a$ = a$ + x$
WEND
```

au cas où la commande INPUT risque de désynchroniser le déroulement de votre programme.

Reprise d'interruption

EI (Enable Interrupt)
EI est le pendant de DI. Nous savons que DI met les interruptions sur "répondeur". La commande EI consiste à exécuter, dans l'ordre d'arrivée et par priorité décroissante, les demandes enregistrées.

EI remet le programme à l'écoute des quatre réveils.

Exemple

```
10 EVERY 50,1 GOSUB 100
20 '
30 WHILE a = 0
40   DI                   ' desactive les reveils
50   INPUT a$ : i=0
60   EI                   ' reactive les reveils
70 WEND
80 '
70 '
100 PRINT i : i=i+1       ' sous-programme d'interruption
110 RETURN
```

RETURN
Le bien connu "RETURN" joue le rôle de la commande EI, lorsqu'il termine un sous-programme d'interruption.

Arrêt d'interruption

ON BREAK STOP
Cette commande annule l'effet de "ON BREAK GOSUB".

Exemple

```
10 ON BREAK GOSUB 100
20 PRINT "le premier break n'arrete pas le programme"
40 GOTO 30
50 '
100 ON BREAK STOP
110 RETURN
```

REMAIN(i) i = Ø, 1, 2 ou 3
A tout instant, le programme peut demander combien il reste de temps avant la prochaine interruption sur un des quatre réveils (Ø à 3).

Une réponse nulle signifie qu'il n'y a ni commande EVERY ni commande AFTER en cours.

La lecture de cette information désactive le réveil i.

SQ(i) i = 1, 2 ou 4
Cette variable contient l'état du canal i (voir le chapitre II).

Sa lecture annule l'effet de la commande ON SQ(i) GOSUB ligne.

END
La fin d'un programme désactive toutes les interruptions en attente mais ne vide pas les files musicales.

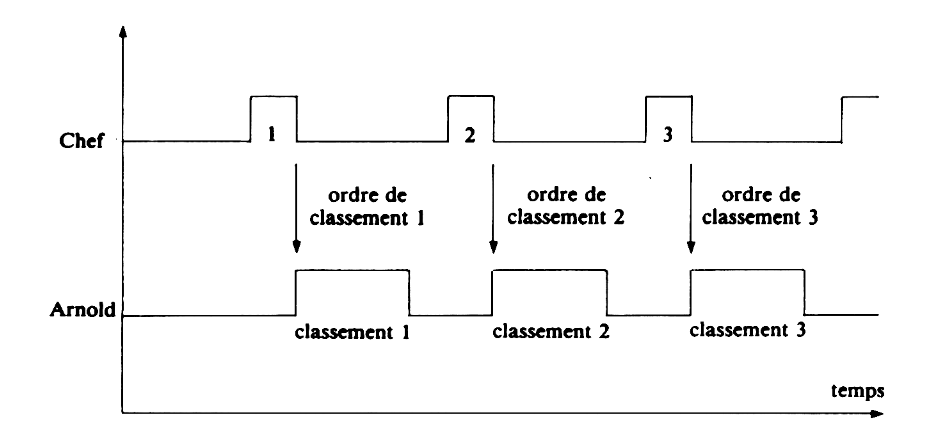
TABLEAU RÉCAPITULATIF SUR LES INTERRUPTIONS

Interruption	Activation	Suspension	Reprise	Arrêt
réveils Ø à 3	EVERY AFTER	DI INPUT	EI (RETURN) fin INPUT	REMAIN END AFTER' EVERY'
musique	ON SQ( )			sous-prog. SQ( ) SOUND END
break	ON BREAK GOSUB			ON BREAK STOP

Voici, pour terminer, une "maxime" et son explication :

Un sous-programme d'interruption doit être le plus court possible.

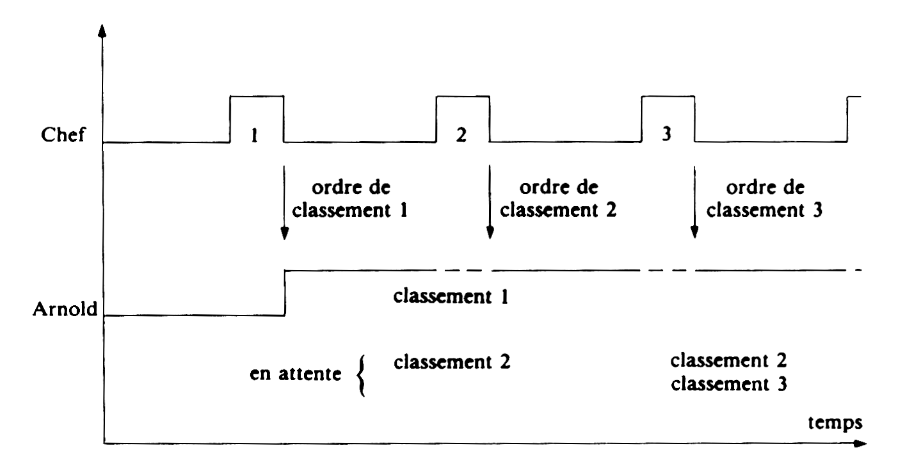
Arnold a un chef maniaque qui l'appelle toutes les dix minutes pour lui demander de classer des documents. Si ce classement se fait en moins de dix minutes, Arnold le fait sans problème.



Arnold a le temps de classer les documents.

Dans ce cas, tout se passe bien car Arnold a le temps de faire son travail.

Mais lorsqu'un classement est plus long, Arnold reçoit un nouvel ordre avant d'avoir pu terminer le précédent. Il est alors obligé de noter qu'il doit faire le second travail et continuer le premier.



Arnold est débordé...

Sur le troisième ordre de classement, Arnold n'a pas encore terminé son premier classement. Il est clair qu'Arnold est vite débordé comme l'est l'Amstrad si les sous-programmes d'interruption sont plus longs que le temps entre deux interruptions.

Il est préférable que le temps d'exécution d'un sous-programme d'interruption soit plus court que le temps entre deux interruptions, surtout si ces interruptions sont périodiques. Cela évitera les deux écueils suivants :

— risque de ne plus avoir de place pour enregistrer les appels,
— impossibilité de travailler en temps réel, comme le montre l'exemple suivant.

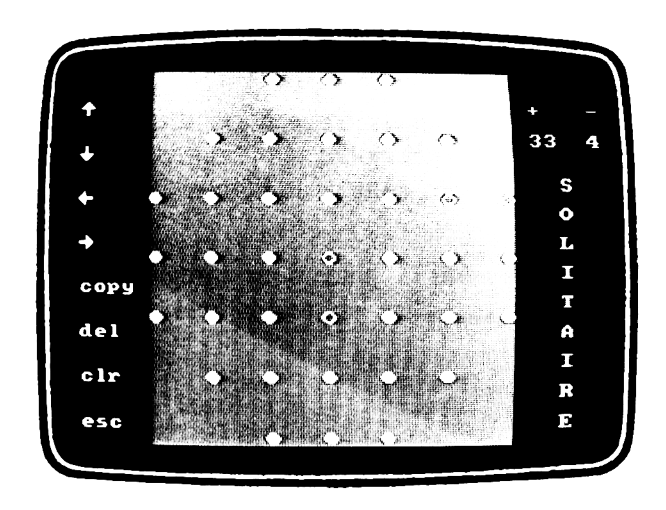
Exemple

```
10 CLS : WINDOW #1,1,16,1,1
20 '
30 EVERY 50 GOSUB 100          ' toutes les secondes
40 '
50 GOTO 50                     ' ou tout autre programme
60 '
100 '                          ' sous-programme : affiche l'heure
110 FOR i=1 TO 10000 : NEXT '    retard
120 '
130 s = s + 1                  '   une seconde supplementaire
140 sec=s MOD 60: mn=(s\60) MOD 60 :heure=(s\3600) MOD 24
150 '
160 PRINT #1,USING "####";heure;mn;sec
170 RETURN
```

La boucle de la ligne 1ØØ retarde l'exécution du sous-programme. Les interruptions arrivent cependant toutes les secondes mais n'activent pas immédiatement le sous-programme. L'heure affichée est fausse.

Vous trouverez dans le programme MACAO des exemples d'utilisation des interruptions, ainsi que dans le programme suivant.

Solitaire



Un damier. Trente-sept pièces à éliminer les unes après les autres. Le joueur enlève d'abord une pièce grâce aux quatre flèches et en appuyant sur la touche [clr]. Il dirige ensuite son curseur sur une pièce qu'il désire bouger. Le choix fait, il valide la pièce en appuyant sur la touche [copy]. Quelques flèches pour arriver sur une case vide et la pièce validée fait un saut jusqu'à cette case, grâce à la touche [del]. Une prise se fait selon les mêmes règles que celles du jeu de dames. La pièce qui se déplace doit atterrir sur une place vide après avoir mangé une pièce et une seule.

Amusez-vous bien et ne craignez plus d'égarer des pions. L'Amstrad sait où ils sont.

```
1 '         **************
2 '         *  SOLITAIRE *
3 '         **************
10 GOSUB 1000              ' initialisation
20 '
30 GOSUB 1500              ' dessin initial
40 '
50 ON BREAK GOSUB 5000     ' sortie elegante
55 '
60 i=2: j=2                ' coord. de depart
70 EVERY 30 GOSUB 2000     ' clignot.point courant
80 '
90 i0=-1 : j0=-1           ' point a deplacer inexistant
100 EVERY 50,1 GOSUB 2200' clignot.point a deplacer
110 '
120 GOSUB 500              ' attente caractere ( a )
130 '
140 WHILE jamais = 0       ' boucle sans fin
145    ii = i :  jj = j            ' pas de modif.avant verif.
150    IF a=240 THEN jj = j  - 1    : GOTO 300     ' haut
160    IF a=241 THEN jj = j  + 1    : GOTO 300     ' bas
170    IF a=242 THEN ii = i  - 1    : GOTO 300     ' gauche
180    IF a=243 THEN ii = i  + 1    : GOTO 300     ' droite
190 '
200    IF a=224 THEN GOSUB 3000 : GOTO 400       ' copy
210 '
220    IF a=127 THEN GOSUB 3500 : GOTO 400       ' del
230 '
240    IF a= 16 THEN GOSUB 4000 : GOTO 400       ' clear
250 '
270 '
300    DI
310    IF (ii+jj<2)OR(ii+jj>10)OR(jj-ii)<-4 OR(jj-ii)>4 GOTO 400
320      i=MAX(0,MIN(6,ii)) : j=MAX(0,MIN(6,jj))   ' point valable
330 '
400    EI :  GOSUB 500      ' attente caractere ( a )
410 '
430 '
440 WEND
450 '
490 '-----------------------------------------------------------------
491 '              " inkey$ " -> a$,a
492 '              ------------------
500 WHILE INKEY$<>"" : WEND        ' vide le buffer
520 a$=""
530 WHILE a$="" : a$=INKEY$ : WEND
550 a=ASC(a$)
560 '
570 RETURN
580 '
590 '
989 '-----------------------------------------------------------------
990 '               initialisation
991 '               --------------
1000 MODE 1
1010 INK 0,0 :INK 1,6         ' 0:noir 1:rouge
1020 INK 2,9 :INK 3,15        ' 2:vert 3:orange
1030 '
```

```
1040 WINDOW #1,7,31,1,25       ' jeu
1050 WINDOW #2,32,40,3,3       ' resultats
1060 WINDOW #3,32,40,5.5       ' chiffres
1070 WINDOW #4.1,6,1,25        ' mode d'emploi
1075 WINDOW #5,35,35,7,25      ' solitaire
1080 BORDER 0
1090 PAPER #1,2:PAPER #2,0
1100 PAPER #3,0:PAPER #4,0:PAPER #5,0
1110 CLS:CLS#1:CLS#2:CLS#3:CLS#4
1120 '
1130 PEN #1,1 :PEN #2,1
1140 PEN #2,2 :PEN #4,3 :PEN #5,3
1150 '
1160 DIM m(6,6)                ' damier
1170 inc = 4                   ' increment
1180 '
1190 reste=37 : enleve =0
1200 GOSUB 1400                ' resultats
1210 '
1220 GOSUB 1450                ' mode d'emploi
1230 '
1300 RETURN
1399 ' ---resultats
1400 PRINT #2," +   -"
1410 PRINT #3,reste;enleve
1420 RETURN
1430 '
1450 PRINT #4,,CHR$(240),,CHR$(241),,CHR$(242),,CHR$(243),
1460 PRINT #4,,"copy",,"del",,"clr",,"esc"
1470 PRINT #5," S O L I T A I R E"
1480 RETURN
1490 '------------------------------------------------------------
1491 '                 dessin initial
1492 '                 --------------
1500 '
1510 FOR j = 0 TO 6
1520   FOR i = 0 TO 6
1530     m(i,j) = 1
1540     IF i+j < 2 THEN m(i,j) = -1   '
1550     IF i+j >10 THEN m(i,j) = -1   '
1560     IF j-i <-4 THEN m(i,j) = -1   '
1570     IF j-i > 4 THEN m(i,j) = -1   '
1580 '
1590     LOCATE #1,1+inc*i,1+inc*j
1600     IF m(i,j) <> -1 THEN PRINT #1,CHR$(230+m(i,j))
1610 '
1620   NEXT i
1630 NEXT j
1640 '
1680 RETURN
1990 '-----------------------------------------------------------
1991 '      IT  clignotement point ordinaire
1992 '          ---------------------------
2000 DI :IF m(i,j) < 0 GOTO 2070
2002  FOR temps = 0 TO 2
2005   LOCATE #1,1+inc*i,1+inc*j
2010   PEN #1,3 : PRINT #1,CHR$(230+m(i,j))
2015  NEXT
2020   LOCATE #1,1+inc*i,1+inc*j
2030   PEN #1,1 : PRINT #1,CHR$(230+m(i,j))
2070 RETURN
2190 '------------------------------------------------------------
2191 '      IT  clignotement point a deplacer
2192 '          ----------------------------
2200 DI :IF i0 < 0 OR j0 < 0 GOTO 2270
```

```
2202  FOR temps = 0 TO 2
2205   LOCATE #1,1+inc*i0,1+inc*j0
2210   PEN #1,2 : PRINT #1,CHR$(230+m(i0,j0))
2215  NEXT
2220   LOCATE #1,1+inc*i0,1+inc*j0
2230   PEN #1,1 : PRINT #1,CHR$(230+m(i0,j0))
2270 RETURN
2990 '-----------------------------------------------------------
2991 '         copy  ( piece a deplacer )
2992 '          ----------------------
3000 '
3010 i0 = i
3020 j0 = j
3030 RETURN
3490 '-------------------------------------------------------------
3491 '            del ( coup a jouer )
3492 '            -------------------
3500 DI
3510 difi = ABS(i-i0) :difj = ABS(j-j0) : difij=difi+difj
3520 IF (difi=0 OR difi=2)AND(difj=0 OR difj=2)AND difij<>o
     THEN ok=1   ELSE ok=0
3530 '
3540 im=(i+i0)/2 : jm=(j+j0)/2           ' point milieu
3550 IF ok=0 OR m(i,j)<>0 OR m(im,jm)<>1
          OR m(i0,j0)<>1 GOTO 3700       ' non vide ou hors ecran
3560 '
3570   ii=i : jj=j
3580   i=im : j=jm :m(i,j)=0:GOSUB 4010 ' efface point milieu
3585   i=i0 : j=j0 :m(i,j)=0:GOSUB 4010 ' efface origine
3590   i=ii : j=jj :m(i,j)=1:GOSUB 4010 ' allume destination
3600   reste = reste - 1
3605   i0 = i : j0 = j                  ' point origine
3610   enleve = enleve + 1
3620   GOSUB 1400                       ' resultats
3640 '
3700 EI : RETURN
3990 '------------------------------------------------------------
3991 '           clear ( enlever une piece de force)
3992 '           --- 4010 : positionne une piece---
4000 DI :IF m(i,j)<>0 THEN reste=reste-1 :enleve=enleve+1
4002 m(i,j) = 0
4005 GOSUB 1400                         ' resultats
4010  LOCATE #1,1+inc*i,1+inc*j
4020   PEN #1,1 : PRINT #1,CHR$(230+m(i,j))
4070 EI :RETURN
4990 '-------------------------------------------------------------
4991 '          break
4992 '          -----
5000 CLS #4 :INPUT #4,"voulez-vous arreter ce jeu";x$
5010 IF LOWER$(x$)<>"o" THEN CLS #4:GOSUB 1450: RETURN
5020 INPUT #4,"voulez-vous un nouveau jeu";x$
5030 IF LOWER$(x$)<>"o" THEN CLEAR : CLS ::END
5040 CLEAR : GOTO 10
5050 '
5060 '  fin
```

Commentaires

Le damier est représenté par une matrice 7 × 7 (ligne 116∅). Le point courant a pour coordonnées i et j, le point à déplacer i∅ et j∅. L'élément m (i,j) vaut 1 si une pièce est sur la position (i,j), ∅ si il n'y a pas de pièce et −1 si cette position est interdite.

Les lignes 7Ø et 1ØØ font clignoter les pièces toutes les 4 dixièmes de secondes. Les sous-programmes d'interruption sont masqués (lignes 2ØØØ et 22ØØ) pour éviter que les variables i, j, iØ, et jØ soient manipulées "en même temps" par plusieurs niveaux d'interruption. Le démasquage est contenu implicitement dans les commandes RETURN (lignes 2Ø7Ø et 227Ø).

Variations

Les maniaques de la simulation pourront créer un bruit de choc pour le déplacement des pièces. Les amateurs de jeu d'action pourront imposer un temps limite avec pénalisation et alarme en cas d'essais interdits (ligne 355Ø). Ils pourront limiter aussi l'utilisation de la touche [clr] qui enlève une pièce lorsque la situation est coincée.

La réalisation d'un jeu de dames à partir du solitaire ne pose pas de problème particulier car la prise des pièces est identique dans les deux jeux.

Chapitre IV
Le clavier

L'Amstrad propose une gestion du clavier extrêmement souple, d'autant plus remarquable que celle-ci est accessible grâce à des instructions BASIC. A côté de la possibilité, désormais classique, de programmer des touches de fonction (KEY), on trouve des instructions permettant de modifier la vitesse de répétition des touches (SPEED KEY), et surtout de **redéfinir** entièrement le clavier (KEY DEF). Cette dernière possibilité, alliée à la redéfinition graphique des caractères (vue au chapitre I), permet à l'utilisateur d'adopter n'importe quel alphabet. Enfin, on trouve une instruction peu courante mais extrêmement intéressante, INKEY (et JOY), qui avec l'instruction INKEY$ permet au programmeur de connaître l'état du clavier à tout instant.

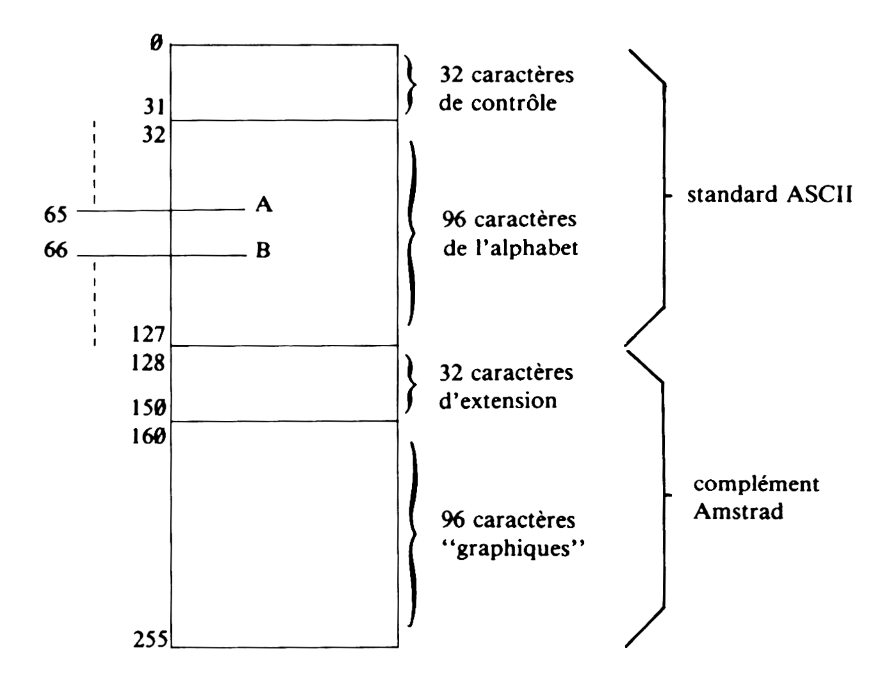
La redéfinition du clavier

Numérotation des caractères

La plupart des micro-ordinateurs adoptent le jeu de caractères ASCII. Chaque lettre, majuscule ou minuscule, chaque chiffre, ainsi que divers signes (+, −, $, &, ., ; etc.), sont associés à un numéro "standard", la valeur ASCII du caractère. Le tableau des valeurs ASCII est donné dans le manuel utilisateur de l'Amstrad. On y retrouve les caractères Ø à 9 numérotés en ASCII de 48 à 57, les majuscules de l'alphabet numérotées de 65 à 9Ø, les minuscules de 97 à 122, etc.

Ce tableau est complété dans l'Amstrad jusqu'à 256 caractères, les caractères suivants étant soit des caractères d'**extension** (voir plus loin), soit des caractères à vocation graphique (dessin d'une note, d'une fusée, d'une figurine, etc.).

Le tableau des caractères se présente ainsi :



Numérotation des touches

Vous trouverez dans le manuel utilisateur le tableau des numéros de touche. Ces numéros permettent de repérer la position d'une touche sur le clavier. Par exemple la touche Ø est la touche qui se trouve le plus en haut à droite du clavier ; elle est marquée ↑. La touche 47 est la grande touche située en bas du clavier, etc.

De la touche au caractère

Sur une machine à écrire mécanique, l'appui d'une touche provoque la frappe sur le papier d'une tige métallique au bout de laquelle est gravée la forme d'un caractère. Si l'on appuie en même temps sur cette touche et sur la touche SHIFT, on déclenche la frappe d'une autre tige métallique. Les caractères gravés sur ces deux tiges sont par exemple une lettre en majuscule et en minuscule. **A une touche est reliée une ou deux tiges** métalliques ; et pour éviter toute confusion, la touche est elle-même marquée du (ou des caractères) gravé(s) sur les tiges qui lui sont associées.

L'Amstrad propose un système identique (à l'électronique près !) dans son principe ; mais à la différence de la machine à écrire, il est possible d'adopter pour chaque touche la ou les tiges de son choix. Pour cela, il n'est plus possible de repérer un couple "touche-tige" par un caractère. C'est là que les numéros de touche et de caractère (tige) interviennent. Il ne faut plus dire : "L'appui de la touche marquée **A** déclenche la frappe de la tige supportant le caractère **A**", mais : "L'appui de la touche numérotée 69 déclenche la frappe de la tige numérotée 97".

L'instruction KEY DEF permet de modifier à loisir cette proposition. Faites par exemple KEY DEF 69,1,98 qui ordonne maintenant que l'appui de la touche 69 déclenche la frappe de la tige 98 (le 1 signifie que la touche peut être répétée), puis frappez la touche 69 (elle est marquée **A**). Le caractère **b** est affiché. C'est le caractère dont la valeur ASCII est 98.

Remarque

L'Amstrad permet à l'utilisateur de redessiner sur chaque tige la forme du caractère. Mais n'utilisez pas cette possibilité pour redéfi-

nir votre clavier. En effet, le BASIC en particulier et les ordinateurs en général ne connaissent ni **a**, ni **b**, ni **&**, ni **+**. Ils ne connaissent que des numéros. Pour le BASIC, **a** est le caractère 97, **b** le caractère 98, etc. Si, au lieu d'utiliser KEY DEF, vous utilisiez la redéfinition graphique des caractères pour donner au caractère inscrit sur la tige 97 la forme d'un **b** , vous afficheriez bien un **b** en appuyant sur la touche marquée **A**, mais le BASIC comprendrait que vous avez utilisé le caractère 97, c'est-à-dire pour lui un "a". Très pratique pour piéger son meilleur ami ; très peu pour écrire un programme !

Nous vous conseillons de respecter le partage des rôles suivant. La redéfinition des touches permet de réorganiser le clavier; on peut ainsi composer un clavier AZERTY, ou alphabétique; améliorer l'ergonomie de certains groupes de touches. La redéfinition graphique des caractères permet, en **respectant** le standard ASCII (par exemple dans une application de traitement de texte avec une imprimante spécialisée), de créer des caractères nouveaux tels que **à, ê, è, ç,** etc., ou bien de redessiner, en particulier pour des jeux, les caractères de n[os] 16Ø à 255.

KEF DEF

Cette instruction permet de modifier les caractères associés à une touche dans les trois cas suivants :

- la touche est appuyée seule,
- la touche est appuyée en même temps que SHIFT,
- la touche est appuyée en même temps que CTRL.

On indique derrière l'instruction le numéro de la touche, puis 1 ou Ø, s'il l'on veut que la touche soit répétée en cas d'appui prolongé de celle-ci, puis les numéros des trois caractères qu'on associe à cette touche.

Dans le programme suivant, on redessine tout d'abord le caractère 16Ø pour obtenir un **ç**. Puis on associe à la touche marquée **C** (numéro 62) les caractères 99 **(c)**, 67 **(C)** et 16Ø c'est-à-dire **ç**. (Les deux premiers caractères sont déjà associés à cette touche à la mise sous tension de l'Amstrad.)

Après l'exécution du programme, vérifiez que la touche **C** est toujours "répétitive" et que l'appui de cette touche avec la touche CTRL affiche bien un **ç**. Vous pouvez enfin écrire "français" correctement !

```
10 SYMBOL AFTER 0
20 SYMBOL 160,0,0,60,102,96,102,60,112 'dessin du caractere 160
30 KEY DEF 62,1,99,67,160          'ctrl+touche C donne c cedille
```

KEY

Il existe 32 caractères d'extension numérotés de 128 à 159. Lorsque l'on affiche ce caractère par un PRINT CHR$( ) la forme graphique qui lui est associée est affichée. En revanche, si l'on frappe une touche associée à ce caractère, une chaîne de caractères en **extension** est générée et affichée. L'instruction KEY permet de définir le contenu de cette chaîne. Elle est suivie du numéro de caractère (entre 128 et 159), puis de la chaîne de caractères.

A la mise sous tension de l'Amstrad, les touches du "pavé numérique" (à droite du clavier principal) sont affectées à des caractères d'extension.

Le programme suivant affecte aux touches 64 et 65 (marquées 1 et 2 en haut du clavier) les caractères d'extension 141 et 142 lorsqu'elles sont appuyées avec la touche CTRL (les autres affectations de ces touches ne sont pas modifiées). Puis il associe aux caractères d'extension 141 et 142 les chaînes de caractères. Vérifiez après avoir exécuté ce programme que la frappe des touches CTRL et 1 provoque le lancement du programme, que celle de CTRL et 2 provoque l'instruction LIST.

```
10 KEY DEF 64,0,49,33,141
20 KEY DEF 65,0,50,34,142
30 KEY 141,"run"+CHR$(13)
40 KEY 142,"list"+CHR$(13)
```

La mémoire du clavier

Lorsque le BASIC exécute un programme, par exemple l'affichage de caractères à l'écran ou sur une imprimante, il ne s'occupe pas de ce que l'utilisateur peut éventuellement frapper au clavier. Ce n'est qu'une fois qu'il a exécuté son programme qu'il vient lire le clavier pour connaître les ordres qui lui seront communiqués. Et pourtant, durant le temps d'exécution d'un programme, les éventuelles frappes de touches sont mémorisées par l'Amstrad.

C'est la faculté des systèmes **en temps réel** de "faire plusieurs choses à la fois". L'Amstrad ne se contente jamais de faire "tourner" un programme. Grâce à un système d'**interruptions**, il partage son temps en fractions très courtes, à gérer l'écran, le générateur de sons, l'horloge interne, le clavier, et à exécuter éventuellement un programme BASIC.

Ainsi, quoiqu'il ait à faire par ailleurs, le système consacre une certaine partie de son temps, tous les 1/5Øe de seconde, à examiner l'état du clavier. Il note les touches pressées et les mémorise dans un **buffer**. Celui-ci est vidé au fur et à mesure que le programme principal (le BASIC) vient lui-même lire ce buffer pour savoir si des touches ont été frappées.

La petite "expérience" suivante vous donnera une idée plus claire de la mémoire du clavier. Faites exécuter à l'Amstrad l'instruction FOR I = Ø TO 2ØØØØ : NEXT et durant l'exécution, écrivez au clavier par exemple la liste des lettres dans l'ordre alphabétique.

Vous devriez avoir terminé avant que l'Amstrad n'ait eu le temps d'exécuter l'instruction. Lorsqu'il en aura fini avec cette boucle, les premières lettres de l'alphabet de A jusqu'à T s'afficheront. Vos vingt premières frappes au clavier ont été mémorisées, les suivantes sont perdues.

Les vingt premières frappes se sont "entassées" dans le buffer des touches, et comme le programme BASIC ne venait pas les lire, il y a eu saturation du buffer. La mémoire du clavier ne dépasse pas en effet vingt frappes. Si vous frappez maintenant d'autres touches, celles-ci sont immédiatement affichées à l'écran. Le BASIC ne fait rien d'autre qu'attendre vos ordres et vide constamment le buffer dès que celui-ci se remplit, dès que vous frappez une touche.

INKEY$ et INKEY

L'instruction a$ = INKEY$ place dans la variable a$ soit le caractère vide “ ”, si le buffer des touches est à ce moment vide, soit le premier caractère placé dans le buffer. Ce premier caractère est associé à la touche la plus anciennement frappée au clavier, que le BASIC n'a pas pris en compte. INKEY$ permet ainsi de perdre le moins de frappes possible.

Cela peut avoir des inconvénients, par exemple dans un programme de ce type :

```
1000 PRINT "voulez-vous continuer?"
1010 a$="": WHILE a$="": a$=INKEY$: WEND
1020 IF a$<>"o" THEN END
```

L'instruction en 1Ø1Ø initialise a$, puis boucle tant que a$ est égal au caractère vide, donc tant que le buffer est vide. Dès que l'utilisateur répond, on sort de la boucle et on analyse sa réponse en 1Ø2Ø. En revanche, si au moment de l'exécution de INKEY$, le buffer des touches n'est pas vide, on sort immédiatement de la boucle avec un caractère placé dans a$, associé à une touche anciennement frappée, qui ne répond pas à la question posée.

Une première solution pour éviter ce problème consiste, avant de poser la question, à vider le buffer des touches, en rajoutant par exemple :

```
990 WHILE INKEY$<>"": WEND
```

L'instruction INKEY peut elle aussi permettre de résoudre ce problème. Cette instruction donne en effet l'état au **moment de l'instruction** de la touche testée. Elle est suivie du numéro de touche. Par exemple **a** = INKEY (n° touche) place dans **a** l'état de la touche au moment de l'instruction :

−1 si la touche n'est pas pressée,
+Ø si elle est pressée seule,
32 si elle est pressée en même temps que SHIFT,
128 si elle est pressée en même temps que CTRL,
16Ø si elle est pressée en même temps que SHIFT et CTRL.

Le programme suivant offre une seconde solution à notre petit problème. On est sûr que la touche marquée O (numéro 34) ou que

la touche marquée N (numéro 46) a été frappée **après** l'affichage de la question. Cependant elle oblige l'utilisateur à frapper soit la lettre O, soit la lettre N et aucune autre, pour sortir de la boucle.

```
1000 PRINT "voulez-vous continuer?"
1010 a=-1: b=-1
1020 WHILE a=-1 AND b=-1
1030   a=INKEY(34): b=INKEY(46)
1040 WEND
1050 IF b<>-1 THEN END
```

Chapitre V

La machine

Le microprocesseur Z8Ø

La description que nous faisons du Z8Ø ne prétend pas être complète. Les lecteurs désireux d'aller plus loin dans la connaissance de ce microprocesseur pourront se reporter à la documentation du constructeur ZILOG, ou aux nombreux livres consacrés au Z8Ø.

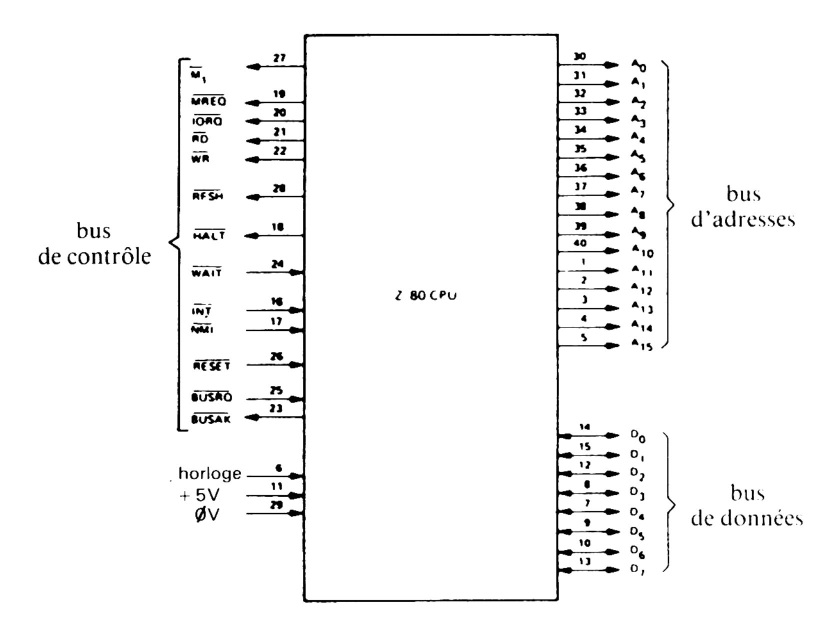


Schéma du brochage du Z8Ø.

Signal d'horloge

L'Amstrad utilise le microprocesseur Z8Ø A, identique au Z8Ø mais pouvant fonctionner plus rapidement. La fréquence du signal d'horloge, qui lui permet de "rythmer" ses opérations, est de 4 MHz. Ceci assure un temps moyen d'instruction d'une microseconde (un millionième de seconde).

Le bus de contrôle

Il comporte l'ensemble des signaux nécessaires au fonctionnement du Z8Ø avec ses voisins. Les fils MREQ et IORQ permettent au Z8Ø de sélectionner pour une lecture (par RD read) ou pour une écriture (par WR write) soit la mémoire, soit les circuits d'entrée/sortie. Par l'intermédiaire de ceux-ci, le microprocesseur peut contrôler les différents périphériques : clavier, écran, générateur de sons, lecteur de cassettes, imprimante, éventuellement lecteur de disquettes.

Notons aussi l'entrée INT (interruption). Cette entrée permet à un circuit externe d'obliger le Z8Ø à interrompre le traitement en cours et à effectuer un traitement "d'interruption". Dans l'Amstrad une horloge externe interrompt le Z8Ø tous les 1/3ØØ^{e} de seconde. Cette interruption permet au Z8Ø de synchroniser ses traitements (gestion de l'écran, du générateur de sons...).

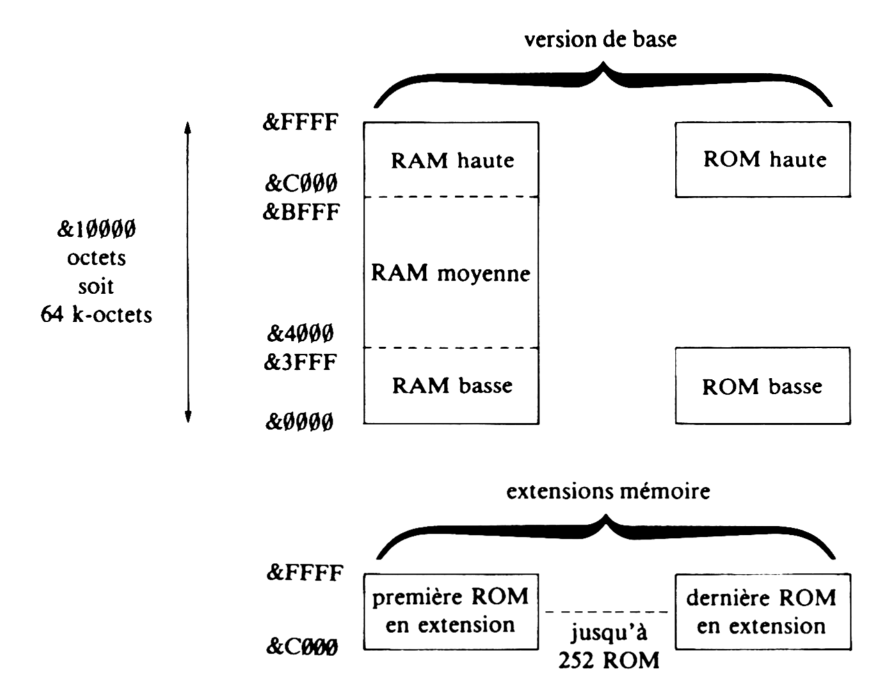
Le bus de données

Sur les huit fils de ce bus circulent les informations échangées entre le Z8Ø et la mémoire ou les circuits d'entrée/sortie. La dimension du bus de données du Z8Ø le fait ranger dans la catégorie des microprocesseurs "8 bits". L'information est organisée en octets, ensembles de huit bits.

Le bus d'adresses

Il comporte seize fils. Le Z8Ø peut donc adresser directement 2^{16} cases mémoire (soit 64 k-octets) et 2^{16} circuits d'entrée/sortie. Nous voyons maintenant comment le Z8Ø de l'Amstrad travaille avec 96 k-octets de mémoire, et comment on peut y ajouter de nombreux k-octets de ROM.

L'organisation de la mémoire de l'Amstrad



En version de base, l'Amstrad comprend :

- 64 k-octets de RAM, placés entre &0000 et &FFFF.
- 32 k-octets de ROM, placés pour la partie basse entre &0000 et &3FFF, pour la partie haute entre &C000 et &FFFF.

En extension, il est possible de rajouter dans la partie haute de la mémoire jusqu'à 252 boîtiers de 16 k-octets de ROM.

Toute opération d'écriture se fait bien évidemment dans la RAM, puisqu'il est impossible d'écrire dans une mémoire ROM. Les choses sont plus compliquées lorsqu'il s'agit de lire une mémoire (sauf lorsque celle-ci se situe entre &4000 et &BFFF, où se trouve uniquement de la RAM). Plusieurs boîtiers mémoire pourraient en effet répondre au Z80, lorsque celui-ci effectue une lecture dans la partie basse ou haute de la mémoire. Un système de sélection des boîtiers mémoire (commandé par un circuit d'entrée/sortie) permet au Z80 de choisir le boîtier mémoire qu'il désire lire ; dans la partie basse soit la RAM basse, soit la ROM basse ; dans la partie haute soit la RAM haute, soit l'une des ROM hautes, repérée par un numéro de 0 à 251.

Contenu de la mémoire en version de base

Adresse	RAM	ROM
&FFFF	Mémoire écran	ROM BASIC
&CØØØ	Pile	
	Données système	
&B1ØØ	Bloc de sauts	
	Données pour ROM haute	
&ABCØ		
&ABFF	MÉMOIRE UTILISATEUR	ROM SYSTÈME
&Ø17Ø	Données pour ROM haute	
&ØØ4Ø	Zone système	
&ØØØØ		

La ROM SYSTÈME et ses zones RAM

La ROM SYSTÈME contient l'ensemble des programmes du système d'exploitation. Elle se compose d'une série de "routines", de programmes, qui organisent le bon fonctionnement de la machine. Les 64 premiers octets (&4Ø) de la ROM SYSTÈME sont recopiés dans la ZONE SYSTÈME de la RAM. Ils sont fondamentaux pour la "navigation dans la mémoire" que nous verrons plus loin.

Les programmes système ont besoin de mémoire vive pour travailler. Une partie de la RAM leur est réservée, les DONNÉES SYSTÈME.

Un programme a aussi besoin d'une PILE pour fonctionner. Cette zone de la mémoire est utilisée pour sauvegarder temporairement des informations essentielles : l'adresse de retour d'un appel à un sous-programme, l'adresse de l'instruction n'ayant pu être exécutée à cause d'une interruption, etc. Cette zone est placée en dessous de l'adresse &C000.
La zone BLOC DE SAUTS contient un ensemble d'instructions de saut à des routines de la ROM SYSTÈME. Nous vous donnerons plus loin une liste d'adresses dans le BLOC DE SAUTS qui vous permettront de faire exécuter très rapidement des actions importantes, telles que lecture d'un caractère au clavier, envoi d'un caractère sur l'écran, etc.

La ROM BASIC et ses zones RAM

La ROM BASIC contient l'interpréteur BASIC. Ce programme est le "chef d'orchestre" de l'ensemble. Il utilise abondamment les programmes de la ROM SYSTÈME pour ses "menus travaux". Les deux zones DONNÉES POUR ROM HAUTE sont réservées pour ses besoins en mémoire vive.

La mémoire écran

Placée entre les adresses &C000 et &FFFF, elle occupe bien les 16 k-octets prévus.

La mémoire utilisateur

Faites le compte, il vous reste 43 664 octets de RAM. Il vous faudra un programme BASIC très important, beaucoup de programmes "machine" et beaucoup de données pour les occuper tous !

Les extensions de ROM

Il est possible de rajouter jusqu'à 252 boîtiers de 16 k-octets de ROM. Ces extensions de mémoire sont de deux types :

Les ROM de premier plan (foreground ROM)

Lorsqu'une ROM de ce type est placée dans la mémoire de l'Amstrad, elle remplace dans le rôle de chef d'orchestre la ROM BASIC.

Il est ainsi possible d'implanter le langage FORTH ou le système CP/M. Des applications telles que traitement de texte, assembleur, debugger, etc., sont (ou seront) disponibles en ROM foreground sur l'Amstrad.

La taille de ces programmes ne sera pas un obstacle à leur intégration sur l'Amstrad. Le système permet en effet à un programme foreground d'occuper quatre ROM foreground (64 k-octets !).

Les ROM de second plan (background ROM)

Ces ROM contiennent des programmes qui sont des extensions :

— du système d'exploitation ; par exemple pour gérer un périphérique supplémentaire (le floppy disk...),
— des programmes foreground ; par exemple une extension des possibilités du BASIC (nouvelles fonctions graphiques...).

L'Amstrad ne peut supporter que sept ROM background au maximum. Cette limite ne doit pas vous effrayer ; il est bien certain que les concepteurs de l'Amstrad ont prévu "très large", que ce soit pour les ROM background, ou pour les ROM foreground. Vous n'utiliserez probablement jamais les 252 ROM possibles !

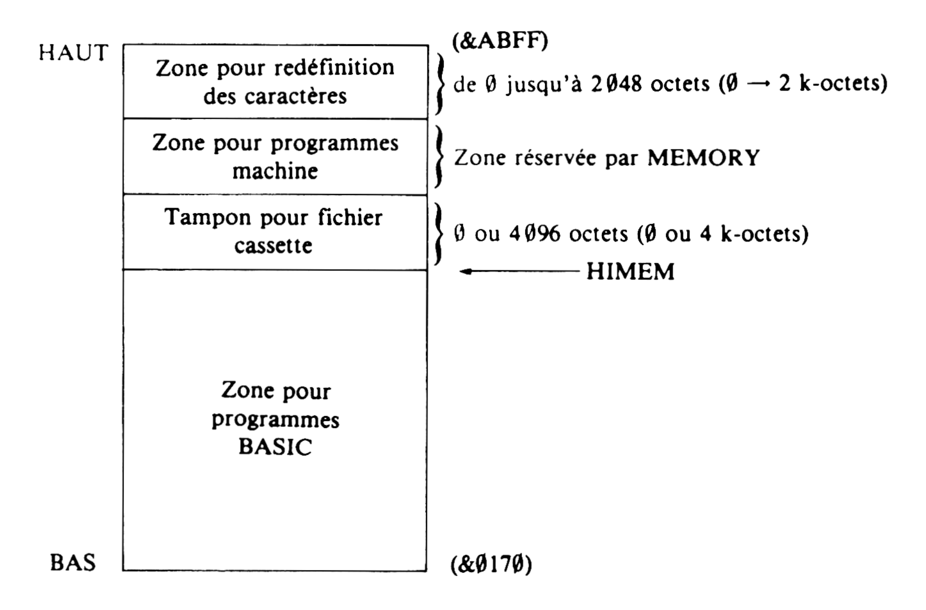
Il faut noter que l'implantation de ROM en extension peut avoir pour effet de diminuer la mémoire utilisateur. A la mise sous tension de l'Amstrad, chaque programme en ROM "se réserve" une zone de données en RAM. La taille des DONNÉES POUR ROM HAUTE peut ainsi augmenter légèrement, au détriment de la mémoire utilisateur.

Retour à la mémoire utilisateur. Les instructions HIMEM et MEMORY

Nous l'avons vu, sans ROM en extension, la zone de &Ø17Ø à &ABFF est réservée à l'utilisateur. Cette zone pourrait être par exemple de &Ø2ØØ à &AØØØ avec des ROM en extension. Disons que la zone utilisateur est comprise entre BAS et HAUT dont les valeurs dans nos exemples seront BAS = &Ø17Ø et HAUT = &ABFF.

L'instruction HIMEM

L'instruction HIMEM permet de connaître l'adresse de l'octet le plus haut de la zone réservée aux programmes BASIC. Au-dessus de cette zone, jusqu'à HAUT, la mémoire utilisateur peut avoir trois utilisations.



L'instruction SYMBOL AFTER (qui réserve une zone pour la redéfinition des caractères), les instructions du type OPEN et CLOSE (ouverture et fermeture de fichiers cassette) ont donc une influence sur la zone réservée au BASIC. Pour vous en convaincre, frappez ces instructions sur votre Amstrad.

Après réinitialisation par CTRL + SHIFT + ESC, faites ?HEX$ (HIMEM). Il s'affiche alors &AB7F (soit HAUT – &8Ø ou &ABFF – &8Ø). A l'initialisation, le BASIC réserve une zone pour redéfinir les seize derniers caractères. Huit octets sont nécessaires pour définir un caractère. La zone réservée est bien de 8 × 16 = 128 = &8Ø octets.

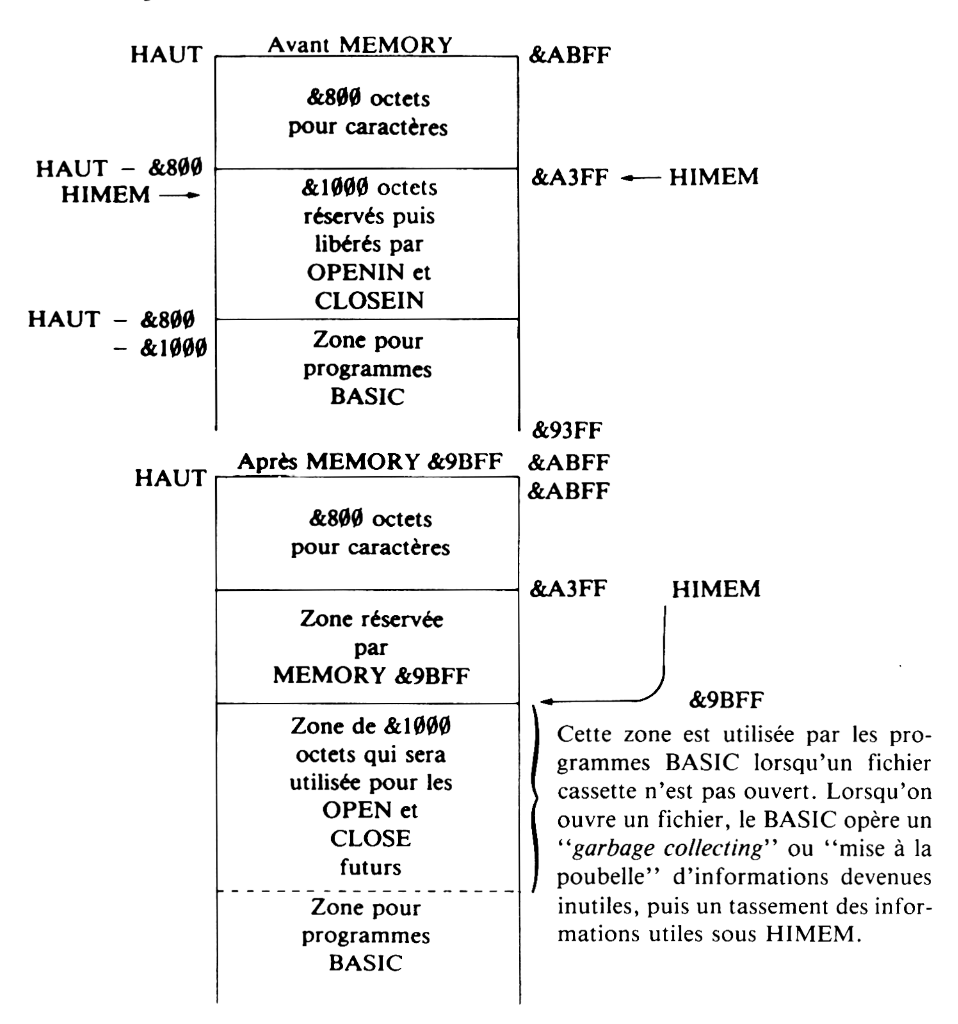
Après SYMBOL AFTER 256, ?HEX$ (HIMEM) affiche &ABFF (= HAUT). Aucune zone "caractère" n'est réservée.

Après SYMBOL AFTER Ø, ?HEX$ (HIMEM) affiche &A3FF (= HAUT – &8ØØ). Une zone de 256 × 8 octets (&8ØØ) est réservée pour redéfinir les 256 caractères.

Faites ensuite OPENIN "TOTO" puis ESC deux fois. Un fichier cassette est ouvert. ?HEX$ (HIMEM) affiche &93FF. 4 K ou &1ØØØ octets supplémentaires ont été réservés par le BASIC pour le fichier cassette.
Faites ensuite CLOSEIN "TOTO" puis ?HEX$ (HIMEM). Vous voyez &A3FF. La zone fichier est libérée.

L'instruction MEMORY

Cette instruction permet de limiter à une adresse précise la zone réservée aux programmes BASIC. Si vous avez suivi notre enchaînement d'instructions jusqu'à maintenant, la zone utilisateur va ainsi évoluer après l'instruction MEMORY &9BFF.



L'instruction MEMORY &9BFF interdit au BASIC de placer ses programmes, ou son tampon pour fichier cassette, au-dessus de &9BFF. La zone de &9BFF à &A3FF est donc libre. Vous pourrez en faire ce que vous voudrez. Par exemple y placer vos programmes machine...

Attention

Le BASIC Amstrad fonctionnera mal si vous augmentez votre zone mémoire "caractère" par SYMBOL AFTER après avoir réservé de la mémoire machine par MEMORY. De même si vous réservez de la mémoire par MEMORY alors qu'un fichier cassette est ouvert. Nous vous conseillons pour tous vos programmes :

— de réserver **tout d'abord,** une fois pour toutes, votre zone pour redéfinir les caractères (utilisez une seule fois l'instruction SYMBOL AFTER),
— **puis** de réserver une fois pour toutes votre zone pour programmes machine (utilisez une seule fois l'instruction MEMORY).

Vous n'aurez alors aucun problème pour ouvrir et fermer vos fichiers cassette, redéfinir vos caractères, rentrer et utiliser vos programmes machine.

Chapitre VI

Le langage machine du Z8Ø

Introduction

Intérêt du langage machine

Pourquoi apprendre un langage complexe et difficile à mettre en œuvre, alors que l'on dispose d'un langage aussi "simple" et puissant que le BASIC ? La réponse est connue. La simplicité (relative !) du BASIC se paie par des temps d'exécution parfois trop longs.

Si l'on programme grâce à l'instruction EVERY un traitement devant s'effectuer toutes les secondes, on voit qu'il est nécessaire que ce traitement dure moins d'une seconde. Le langage machine pourra venir suppléer le BASIC, là où le temps d'exécution est crucial.

Qu'est-ce que le langage machine ?

L'idée géniale, qui est à l'origine des ordinateurs, a été d'associer à chaque opération (ou instruction) qu'on veut lui faire exécuter un nombre. On pourrait ainsi imaginer que pour un ordinateur :

Ø1 signifie additionner,
Ø2 soustraire, etc.

Connaître le langage machine d'un microprocesseur, c'est connaître l'ensemble des instructions qu'il est capable d'effectuer, et savoir pour chaque instruction quel est le nombre, le **code opératoire,** qui lui est associé.

Le langage assembleur

Il est plus agréable de représenter chaque instruction par un ''mnémonique'' :

ADD pour additionner,
SUB pour soustraire, etc.

Cette représentation permet au programmeur de mieux lire son programme jusqu'à ce qu'il le traduise en code machine et le place en mémoire. Nous l'utiliserons pour vous décrire les instructions du Z8Ø.

Au moment où nous écrivons, Amstrad a annoncé la sortie prochaine d'un assembleur. Un assembleur permet d'écrire un programme en représentation mnémonique (langage assembleur) et se charge de la traduction en code machine.

Les registres du Z8Ø

Les registres sont des zones de mémoire vive interne au microprocesseur. Nous vous citons, pour information, les deux registres ''16 bits'' les plus importants qui contrôlent le bon déroulement des programmes.

— le registre PC (Program Counter ou pointeur d'instruction) qui maintient à tout instant l'adresse de l'instruction en cours d'exécution,
— le registre SP (Stack Pointer ou pointeur de pile) qui maintient à tout instant l'adresse du sommet de la pile.

Ces registres sont gérés par le Z8Ø et vous pourrez ne pas vous en occuper. Voici un tableau des autres registres du Z8Ø.

8 bits

A	F	A'	F'
B	C	B'	C'
D	E	D'	E'
H	L	H'	L'
	IX		
	JY		

16 bits

Le Z8∅ possède deux jeux de huit registres de huit bits : le jeu principal (A, F, B, C, D, E, H, L) et le jeu secondaire (A', F', B', C', D', E', H', L'). A un moment donné, seul un jeu est disponible. Des instructions d'échange permettent de sélectionner chaque jeu. **Dans l'Amstrad, le jeu secondaire est réservé au système. Il est donc fortement conseillé de n'utiliser que le jeu principal.**

Le registre F est un registre spécial. Il contient six bits indicateurs (ou flags) positionnés automatiquement par le Z8∅ lors de certaines instructions (deux bits sont inutilisés). Il n'existe pas d'instruction permettant de modifier directement son contenu. Le bit ∅ de ce registre est l'indicateur de retenue (CARRY FLAG). Ce flag est très important dans le système de l'Amstrad car **toutes les routines système dont nous vous donnerons les adresses fournissent en sortie le Carry Flag à 1, si la routine s'est correctement exécutée, à ∅ sinon.**

Tous les autres registres peuvent être considérés comme de la mémoire vive supplémentaire, dans la mesure où des instructions permettent de modifier leur contenu (écrire) ou de transférer ce contenu "ailleurs" (lire). (Bien sûr, d'autres instructions les rendent "supérieurs" à la mémoire vive externe.)

Le registre A est aussi appelé l'accumulateur. C'est sur ce registre que le plus grand nombre d'instructions est possible.

Les registres B, C, D, E, H, L peuvent former trois paires de registres, BC, DE et HL, utilisables comme registres 16 bits.

Signalons, en guise de prélude à la description des instructions, les "spécialités" importantes de certains registres 16 bits.

HL, IX et IY sont très utiles pour accéder à la mémoire.

BC est nécessaire pour accéder aux circuits d'entrée/sortie.

Outils : l'instruction CALL et un programme... BASIC

Avant d'aller plus loin dans l'étude du Z8Ø, nous en savons maintenant suffisamment pour présenter l'instruction CALL. Nous vous proposerons ensuite un petit programme BASIC qui vous permettra de rentrer facilement en mémoire les programmes machine que nous vous donnerons en exemple.

L'instruction BASIC CALL,
ou comment faire exécuter un programme machine.

Le BASIC Amstrad présente l'instruction CALL qui peut sembler, a priori, être l'équivalente de l'instruction EXEC que l'on retrouve dans d'autres BASIC. En réalité, cette instruction se révèle extrêmement plus puissante.

Nous citons pour information la possibilité de faire exécuter une "commande externe" en faisant, par exemple,

CALL | CERCLE, paramètres.

La barre verticale (|) signale que le programme CERCLE (appelé pour tracer un cercle dont l'origine et le rayon sont donnés par les paramètres) se trouve en ROM d'extension. L'organisation interne des ROM permet au système de retrouver l'adresse du programme CERCLE et de le faire exécuter.

Nous nous attacherons uniquement à l'instruction CALL du type CALL &6ØØØ.

Cette instruction permet d'appeler dans un programme BASIC un sous-programme écrit en langage machine (placé ici à l'adresse &6ØØØ).

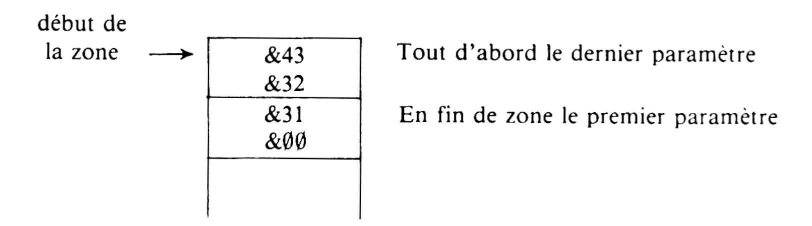
Cette instruction peut, elle aussi, être accompagnée de paramètres.

Exemple

CALL &6ØØØ, &ØØ31, &3243

Voici alors exactement ce qu'exécute l'instruction CALL :

— elle place dans l'accumulateur A le nombre de paramètres contenus dans l'instruction. Ici il y en a 2, &ØØ31 et &3243. Le contenu de A est donc 2;
— elle place ensuite les paramètres dans une zone de données;



Les paramètres, comme on le voit sur la figure ci-dessus, peuvent prendre les valeurs de &ØØØØ à &FFFF. Les poids faibles sont placés avant les poids forts. (On a en mémoire 843 puis &32 pour représenter le nombre &3243.)

— elle place dans le registre IX l'adresse du début de la zone. (Cette zone est placée dans les DONNÉES POUR ROM HAUTE.);
— elle appelle ensuite le programme, commençant ici à l'adresse &6ØØØ.

Tout ceci peut sembler a priori bien complexe. Nous verrons plus loin des exemples très précis d'utilisation de l'instruction CALL avec "passage de paramètres".

Programme d'entrée de données machine

Ce programme vous demande l'adresse du début de la zone mémoire où vous voulez écrire. Ensuite il affiche, adresse après adresse, la valeur de l'adresse et le contenu de la mémoire à cette adresse. Si vous voulez modifier ce contenu, vous indiquez la nouvelle valeur, sinon vous frappez RETURN. Si vous voulez sortir du programme, vous frappez X. Le programme vous propose alors une vérification, qui vous permet de relire (et de modifier à nouveau par le même principe) les valeurs que vous avez introduites. Les adresses et

valeurs doivent être données en hexadécimal. Il ne faut cependant pas indiquer le caractère "&". Le sous-programme placé en 320 effectue la conversion nécessaire.

```
10 ' entree des programmes
20 '
30 MEMORY &05FFF              'reservation memoire
40 INPUT "adresse";A$    'adresse debut
50 GOSUB 320: ADRESSE=A 'conversion en hexa
70 '
80 '
90 ' boucle
100 GOSUB 160      'affiche adresse et ancien contenu
110 GOSUB 220      'traitement demande
120 ADRESSE=ADRESSE+1: GOTO 100
140 '
150 '
160 'affichage adresse et ancienne valeur
170 A=PEEK(ADRESSE)
180 PRINT HEX$(ADRESSE)", ";: PRINT HEX$(A)" ";
190 RETURN
200 '
210 '
220 'traitement demande
230 INPUT "ok";A$
240 IF A$="" THEN RETURN          'pas de modification
250 IF A$="x" THEN GOTO 390       'fin sur X
260 'on prend en compte la valeur fournie
270 GOSUB 320: VALEUR=A           'conversion
280 'IF VALEUR>&FF THEN GOTO 230  'verification
290 POKE ADRESSE,VALEUR: RETURN   'mise en memoire
300 '
310 '280
320 'conversion chaine caracteres en valeur hexa
330 'en entree a$, en sortie a
340 A=VAL("&"+A$)
350 IF A<0 THEN A=A+2^16
360 RETURN
370 '
380 '
390 'fin. demande de verification
400 INPUT "verification";A$
410 IF A$="O" THEN GOTO 40: END
```

Première routine système

Dernier outil avant d'aller plus loin, voici une routine système que nous utiliserons pour visualiser sur l'écran les résultats de nos programmes. Nous vous la présentons sous le format que nous avons adopté pour toutes les autres routines.

Action : Écriture d'un caractère sur l'écran.
Adresse : & BB5D.
Entrée : L'accumulateur A contient le caractère à écrire.
Sortie : Les registres du jeu principal sont modifiés.

L'adresse est l'adresse de départ de la routine. (Elle se trouve bien dans la zone BLOC DE SAUTS.)

L'entrée donne les conditions qu'il faut remplir pour que l'action soit correctement exécutée.

La sortie indique quels sont les registres modifiés. Faites bien attention à ce renseignement ; vous pourriez avoir des surprises désagréables sinon ! (On voit ici, par exemple, qu'après l'exécution de l'affichage, l'accumulateur A ne *contient plus* le caractère.)

L'instruction de chargement LOAD

Définition

L'instruction LOAD (mnémonique LD) permet de charger (de placer) une valeur dans un registre ou dans une case de la mémoire externe. Cette valeur peut être précisée dans l'instruction (valeur immédiate) ou être contenue dans un registre ou une case mémoire.

Oublions un instant les chargements concernant la mémoire et prenons ces deux exemples :

LD A, &Ø7	place dans l'accumulateur A la valeur "immédiate" &Ø7.
LD C, B	place dans le registre C, le contenu du registre B. (Si B contient &15, alors C contient maintenant, lui aussi &15.)

On remarque au passage que dans la représentation mnémonique des instructions, **le destinataire se trouve à gauche de la virgule, la source à droite.**

Programme : Afficher le caractère Ø sur l'écran

Nous utilisons la routine d'affichage en &BB5D. Il faut donc placer dans l'accumulateur A la valeur ASCII du caractère Ø, qui est &3Ø, puis sauter à l'adresse de la routine.

(Les valeurs ASCII des caractères sont données dans l'appendice III du manuel utilisateur.)

Instruction	Code	
LD A, &3Ø	**&3E**	
	&3Ø	valeur
JP &BB5D	**&C3**	
	&5D	adresse
	&BB	

(Vous trouverez en annexe la liste complète des instructions du Z8Ø avec leur code machine.)

JP (jump) est une instruction de saut (équivalente au GOTO du BASIC).

Dans chaque code machine des instructions, nous avons écrit en caractères gras le code opératoire. A la suite, se trouvent les "opérandes" : la valeur à charger dans A, l'adresse où sauter. Vous remarquez que l'on place en mémoire tout d'abord &5D puis &BB, au lieu de &BB puis &5D. **En mémoire, on place tout d'abord les poids faibles, puis les poids forts d'un opérande de deux octets.**

Rentrez, grâce à notre programme d'entrée de données, les valeurs &3E, &3Ø, &C3, &5D, &BB à partir de l'adresse &6ØØØ. N'indiquez pas le "&", la conversion est effectuée par le programme. N'hésitez pas à vérifier vos valeurs.

Frappez ensuite CALL &6ØØØ (indiquez "&") pour faire exécuter ce programme. Le Ø s'affiche.

Adressage de la mémoire

Nous revenons maintenant aux chargements concernant la mémoire.

Pour indiquer l'adresse de la mémoire concernée, on dispose de deux possibilités :

— indiquer "immédiatement" cette adresse ;

LD (&Ø17Ø), D charge dans la mémoire d'adresse &Ø17Ø, le contenu de D.

— placer cette adresse dans l'un des trois registres HL, IX ou IY.

LD B, (HL) charge dans B le contenu de la mémoire "pointée" par HL. Si HL contient la valeur &Ø17Ø, cette instruction placera dans B le contenu de la mémoire d'adresse &Ø17Ø.

Vous avez remarqué que **l'expression (XX) signifie "contenu de la mémoire d'adresse XX"**.

L'utilisation de IX et IY dans le second type d'adressage est légèrement différente de celle de HL. Avec IX et IY, il est possible de rajouter un déplacement "immédiat" à l'adresse contenue dans le registre. Ce déplacement est fourni en complément à deux sur un octet.

LD B, (IX + &Ø3) — place dans B le contenu de l'adresse donnée par la valeur de IX + &Ø3. Si IX = &Ø17Ø, on place dans B le contenu de l'adresse &Ø173.

LD (IY + &FD), E — place le contenu de E dans la mémoire dont l'adresse est égale à la valeur de IY − &Ø3 (complément à deux de &FD). Si IY = &Ø17Ø, on place E dans la mémoire d'adresse &Ø16D.

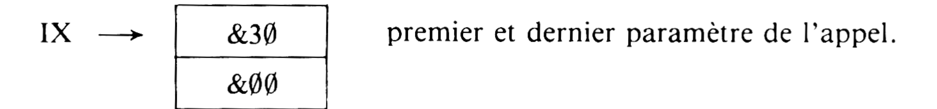
(Pour les lecteurs non familiarisés avec le complément à deux, nous donnons en annexe la méthode de calcul de ce complément.)

Programme : afficher un caractère quelconque
Utilisation de l'instruction CALL avec un paramètre.

Nous allons écrire un programme machine à partir de l'adresse &6Ø1Ø. Ce programme affichera le caractère dont la valeur ASCII sera le paramètre de l'instruction CALL.

CALL &6Ø1Ø, &3Ø — affichera Ø à l'écran.

Nous avons vu que l'instruction CALL place en mémoire pointée par le registre IX le ou les paramètres suivant CALL. Ici nous aurons.



Le programme doit lire ce paramètre et le placer dans l'accumulateur (condition d'entrée de la routine d'affichage). On utilise la deuxième possibilité d'adressage de la mémoire.

Instruction	Code	
LD A, (IX + Ø)	**&DD**	
	&7E	
	&ØØ	déplacement
JP &BB5D	**&C3**	
	&5D	adresse
	&BB	

(Vous remarquez que le code opératoire de LD A, (IX + Ø) utilise deux octets, ce qui est systématique lorsque le registre IX est utilisé.)

Après avoir rentré ce programme en mémoire à partir de &6Ø1Ø, faites :

CALL &6Ø1Ø, &3Ø puis
CALL &6Ø1Ø, &31 puis
CALL &6Ø1Ø, &41.

A l'écran s'affichent Ø, 1 puis A (dont les valeurs ASCII sont &3Ø, &31, &41). Profitez-en pour afficher les caractères "invisibles" de &ØØ à &1F.

Les règles d'adressage pour l'instruction LD

— Tous les chargements entre registres 8 ou 16 bits sont possibles.
LD A, E
LD BC, HL etc.

— Les chargements de mémoire à mémoire sont impossibles.
LD (&Ø17Ø), (HL) est interdit. Il faut faire, par exemple,
LD A, (HL) puis LD (&Ø17Ø), A.

— Les chargements immédiats sont possibles, sauf lorsque deux valeurs immédiates sont présentes dans l'instruction (mais voir aussi la règle suivante).

LD A, &22	
LD BC, &2222	
LD (HL), &22	sont possibles
LD (&Ø17Ø), &2222	est impossible

— Il est possible d'adresser la mémoire en 16 bits lorsque l'adresse est indiquée immédiatement, pas lorsqu'elle est pointée par un registre.

LD (HL), &2222	
LD DE, (HL)	sont impossibles
LD (&2222), DE	est possible

Sauts, appels et retours
Les instructions les plus importantes

Sauts, appels et retours de sous-programme conditionnels

Nous avons déjà rencontré le saut JP.

JP "adresse" permet de sauter à l'"adresse". C'est l'équivalent du GOTO en BASIC.

CALL "adresse" permet d'appeler un sous-programme placé à partir de l'"adresse". C'est l'équivalent du GOSUB en BASIC. (Ne confondez pas le CALL machine et le CALL BASIC !)

Un sous-programme doit obligatoirement se terminer par une instruction de retour de sous-programme du type RET, qui est l'équivalent du RETURN en BASIC.

Sauts, appels et retours conditionnels

Comme leurs noms l'indiquent, les sauts et appels conditionnels permettent de sauter à (ou d'appeler) une adresse si une condition est réalisée. Nous avons retenu les instructions suivantes, qui sont les plus importantes.

JP C,"adresse" saute à l'"adresse" si le Carry Flag est à 1.
JP NC,"adresse" saute à l'"adresse" si le Carry Flag est à Ø.
JP Z,"adresse" saute à l'"adresse" si le Zero Flag est à 1.
JP NZ,"adresse" saute à l'"adresse" si le Zero Flag est à Ø (nous verrons plus loin le Zero Flag (ou Flag Z), appartenant au registre F).

Dans tous les cas, si la condition n'est pas réalisée, le Z8Ø exécute l'instruction qui suit le JP.

Les instructions CALL C, "adresse", CALL NC, "adresse", CALL Z, "adresse" et CALL NZ, "adresse" appellent le sous-programme commençant à l'"adresse", si le flag choisi a la valeur voulue.

Les instructions RET C, RET NC, RET Z, RET NZ effectuent le retour de sous-programme à la condition voulue.

Nous allons voir un exemple d'utilisation de ces instructions après avoir introduit une seconde routine système.

Routine de lecture de caractère sur le clavier

Action : Fournit le premier caractère du buffer des touches du clavier, si celui-ci n'est pas vide.
Adresse : & BB∅9
Entrée : Pas de conditions.
Sortie : S'il y a un caractère, il est contenu dans A et le Carry Flag est à 1.
S'il n'y a pas de caractère, le Carry Flag est à ∅, A est modifié.
Dans tous les cas, les autres registres sont préservés.

Programme : lire le clavier et afficher les touches appuyées

Écrivons une routine qui attend que des touches soient frappées au clavier et qui les imprime aussitôt. La frappe de la touche ESC permet de sortir du programme. Grâce aux instructions de sauts, appels et retours conditionnels, ce programme est extrêmement simple.

Il peut s'analyser ainsi :
- Lire une touche au clavier.
- Si - une touche est pressée.
- Alors - Si - Cette touche est ESC.
    - Alors - Fin.
    - Sinon - afficher le caractère.
    - Boucler.

Voici le programme que nous placerons à l'adresse &6∅2∅ :

Adresse	Instruction	Code
&6∅2∅	CALL &BB∅9	**&CD**
		&∅9 adresse
		&BB

&6Ø23	JP NC, &6Ø2Ø	**&D2**	
		&2Ø	adresse
		&6Ø	
&6Ø26	CP &EF	**&FE**	
		&EF	valeur
&6Ø28	RET Z	**&C8**	
&6Ø29	CALL &BB5D	**&CD**	
		&5D	adresse
		&BB	
&6Ø2C	JP &6Ø2Ø	**&C3**	
		&2Ø	adresse
		&6Ø	

La première instruction appelle le programme de lecture du clavier. La seconde instruction saute en &6Ø2Ø si le Carry est à Ø (NC), donc, s'il n'y a pas de caractère. La troisième instruction, CP, est une instruction de comparaison. La valeur indiquée est comparée avec l'accumulateur A. S'il y a égalité, le Z8Ø positionne le Zero Flag à 1, à Ø sinon. Ici, on compare l'accumulateur A, qui contient le caractère lu au clavier, avec la valeur ASCII de ESC qui est &EF.

L'instruction RET Z effectue un retour de sous-programme si Z = 1, donc si la touche frappée est ESC.

Toutes les conditions sont réunies ensuite, une touche a été frappée, ce n'est pas ESC, pour afficher le caractère. Par chance, il est contenu dans A (condition de sortie de la routine de lecture du clavier). Or, justement, la routine d'affichage affiche le caractère contenu dans A ; il n'est donc pas nécessaire d'utiliser une instruction de transfert. Il suffit d'appeler la routine d'affichage ; puis de boucler.

Après avoir rentré ce programme, faites CALL &6Ø2Ø et pianotez au clavier. Les touches pressées sont affichées à l'écran. N'oubliez pas d'essayer les touches de fonctions, puis faites ESC pour sortir.

Lexique des instructions les plus importantes du Z8Ø

Nous vous conseillons de vous reporter, pour chaque type d'instruction, à l'annexe donnant la liste complète des instructions du Z8Ø.

Vous pourrez ainsi savoir exactement quels sont les registres ou adressages possibles pour chaque instruction.

Remarque : Lorsque nous dirons qu'une instruction est "utilisable comme une autre", cela ne signifiera pas que les deux instructions font les mêmes opérations, mais qu'elles peuvent être exécutées sur les mêmes "objets" (registre, mémoire...).

ADD. Addition

— Addition 8 bits.
ADD permet d'additionner à l'accumulateur A, un registre, un octet mémoire ou une valeur indiquée immédiatement. Le résultat est placé dans l'accumulateur.

ADD B de code **&8Ø**, place dans A la somme de A et B.

ADD &1C de code **&C6** &1C, place dans A la somme de A avec la valeur &1C.

— Addition 16 bits.
Certaines additions entre registres 16 bits sont possibles (voir annexe).

ADD IX, BC de code **&DD &Ø9** place dans IX la somme des contenus de IX et BC.

AND. ET logique

Le AND s'effectue obligatoirement en 8 bits, entre l'accumulateur A et un registre ou un octet mémoire ou une valeur immédiate. Le résultat est placé dans A.
AND (IX + &Ø5) de code **&DD &A6** &Ø5 effectue dans A le ET logique de A avec l'octet pointé par IX + &Ø5.

BIT. Test de bit

Cette instruction permet de tester un bit d'un registre 8 bits ou d'un octet en mémoire.
BIT 2, (HL) de code **&CD &56** permet de tester le bit 2 de l'octet mémoire pointé par HL. Cette instruction positionne le flag Z à 1 si le bit testé est à Ø, à Ø si le bit testé est à 1. (Le bit testé n'est pas modifié.)

Exemple :

```
BIT 3, A
JP Z, & 6ØØØ
```

saute en &6ØØØ si Z = 1, donc si le bit 3 de A est à Ø.

CALL. Appeler un sous-programme

Nous avons vu plus haut les appels de sous-programmes conditionnels ou non.

CP. Comparaison

Cette instruction permet de comparer l'accumulateur A avec un registre, un octet mémoire ou une valeur immédiate. Elle ne modifie ni A, ni le registre ou l'octet comparé. Elle positionne ainsi les flags C et Z.

Z = 1 si A = valeur comparée (Z = Ø sinon).
C = 1 si A est inférieur à la valeur.
C = Ø si A est supérieur ou égal à la valeur.

Exemple :
CP E de code **&BB**
JP C, &6ØØØ
saute en &6ØØØ si C = 1, donc si le contenu de A est inférieur à celui de E.

DEC. Décrémenter (enlever 1)

Cette instruction permet de décrémenter un registre 8 ou 16 bits ou un octet en mémoire.
DEC (HL) de code **&35** décrémente l'octet mémoire pointé par HL.
DEC IX de code **&DD &2B** décrémente le registre IX.

IN. Entrer

Cette instruction permet de lire le contenu d'un circuit d'entrée/sortie. Dans l'Amstrad nous conseillons d'utiliser uniquement l'adressage "par le registre C". Cette instruction permet de placer dans un registre 8 bits le contenu du circuit d'entrée/sortie d'adresse BC.
IN E, (C) de code **&ED &58** place dans E le contenu du circuit d'E/S dont l'adresse est contenue dans BC.

INC. Incrémenter (ajouter 1)

Cette instruction, utilisable comme DEC, permet d'incrémenter un registre 8 ou 16 bits ou un octet mémoire.

JP. Sauter

Nous avons vu les instructions de sauts conditionnels ou non plus haut.

LD. Charger

Nous avons vu cette instruction plus haut. Consultez l'annexe pour connaître exactement tous les adressages possibles.

OR. OU logique

Cette instruction, utilisable comme AND, effectue un OU logique entre A et un registre, ou un octet mémoire, ou une valeur immédiate. Le résultat est placé dans A.
OR &8Ø de code **&F6** &8Ø, place dans A, le OU logique de A avec la valeur immédiate &8Ø.

OUT. Sortir

Cette instruction est utilisable comme IN. Elle permet de placer le contenu d'un registre dans un circuit d'entrée/sortie.
OUT (C), H de code **&ED &61** place dans le circuit d'E/S d'adresse contenue dans BC, le contenu de H.

Les instructions PUSH (pousser ou empiler) et POP (tirer ou dépiler)

Ces instructions permettent d'empiler (PUSH) ou de dépiler (POP) les registres 16 bits AF, BC, DE, HL, IX et IY.

Ces instructions sont très utiles pour sauvegarder les registres dans la pile lors de l'appel d'une routine système, puis pour les restituer après l'exécution.

Exemple :
La routine système d'affichage sur l'écran modifie tous les registres du jeu principal. On veut afficher un caractère à l'écran, sans que les registres HL et DE soient modifiés. Il suffit de faire :

PUSH HL	de code **&E5**	empile HL
PUSH DE	de code **&D5**	empile DE
CALL &BB5D		appelle la routine d'affichage
POP DE	de code **&D1**	dépile DE
POP HL	de code **&E1**	dépile HL

HL et DE ont maintenant le même contenu qu'avant l'appel de la routine d'affichage.

Remarquez l'ordre dans lequel HL et DE sont "pushés" puis "popés". Ce n'est pas le même.

Imaginez que la pile est une pile d'assiettes. HL contient du fromage, DE contient des légumes. Pour sauvegarder HL, on prend une assiette dans laquelle on place HL (le fromage). On place l'assiette sur le dessus de la pile. De même pour sauvegarder DE on place une assiette de légumes sur le dessus de la pile, donc par-dessus l'assiette de fromages.

Pour dépiler, la première assiette que l'on pourra vider sera la dernière placée, donc l'assiette de légumes que l'on restituera dans DE, puis l'assiette de fromages dans HL. En conclusion :
— on dépile les registres dans le sens inverse où on les a empilés,
— le nombre de dépilements doit être exactement égal à celui des empilements. Sinon gare aux indigestions !

RES. Placer à Ø un bit (reset)

Cette instruction est utilisable comme BIT. Elle permet de forcer à Ø un bit d'un registre ou d'un octet mémoire.

RES 7, D de code **&CB &BB** force à Ø le bit 7 de D.

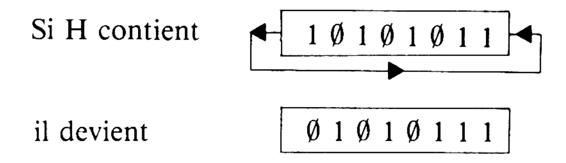
RET. Retour de sous-programme

Nous avons vu plus haut ces instructions. Notez l'instruction RETI utilisée pour le retour d'une interruption.

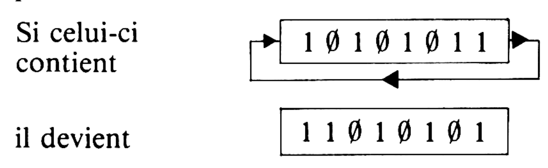
Les instructions RLC (rotation à gauche) et RRC (rotation à droite)

Ces instructions permettent de "faire tourner" un registre 8 bits ou un octet mémoire.

RLC H de code **&CB &14** fait tourner H à gauche.



RRC (HL) de code **&CB &ØE** fait tourner à droite l'octet pointé par HL.



RST. Voir chapitre suivant.

SET. Placer un bit à 1.

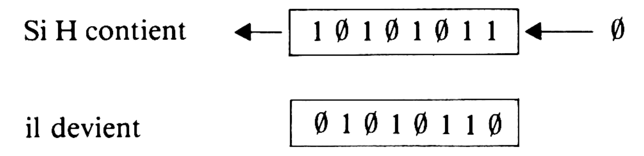
Cette instruction est utilisable comme BIT et RES. Elle permet de forcer à 1 un bit d'un registre 8 bits ou d'un octet mémoire.

SET 1, B de code **&CB &C8** force à 1 le bit 1 du registre B.

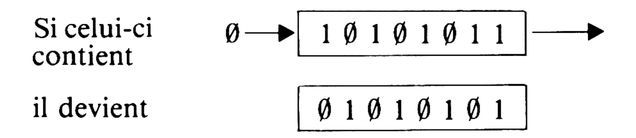
Les instructions SLA (décalage à gauche) et SRL (décalage à droite).

Ces instructions sont utilisables comme RLC et RRC. Elles décalent un registre 8 bits ou un octet mémoire.

SLA H de code **&C8 &24** décale à gauche H.



SRL (HL) de code **&CB &2E** décale à droite l'octet pointé par HL.



SUB. Soustraire

Cette instruction permet de soustraire à l'accumulateur A un registre, un octet mémoire ou une valeur immédiate. Contrairement à l'instruction ADD, aucune soustraction entre registres de 16 bits n'est possible.
SUB (HL), de code **&96**, place dans A le résultat de la soustraction de A avec l'octet mémoire pointé par HL.

SUB&1B, de code **&D6&1B**, place dans A le résultat de la soustraction de A avec la valeur immédiate &1B.

XOR. OU exclusif

Cette instruction est utilisable comme AND et OR. Elle permet d'effectuer un OU exclusif entre l'accumulateur et un registre ou un octet mémoire, ou une valeur immédiate. Le résultat est placé dans A.

XOR C de code **&A9** place dans A le OU exclusif de A avec C.

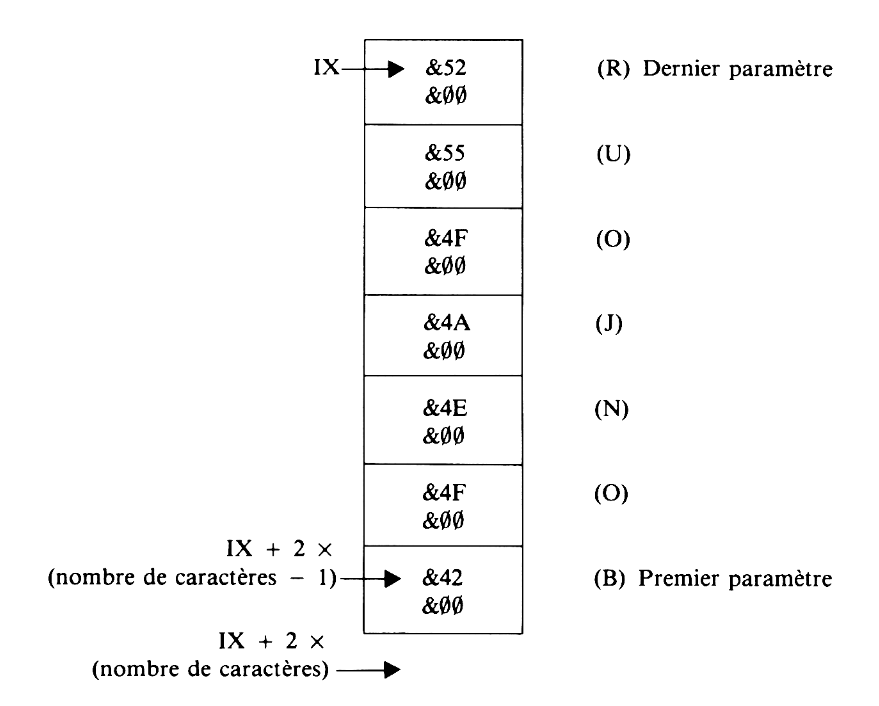
Programme : Affichage de plusieurs caractères à l'écran. Utilisation de l'instruction (BASIC) CALL avec plusieurs paramètres

Ce programme affiche à l'écran les caractères placés comme paramètres de l'instruction BASIC. Nous l'écrirons à l'adresse &61ØØ.

Pour afficher BONJOUR, il faut faire :

```
CALL &61ØØ, &42, &4F, &4E, &4A, &4F, &55, &52
             B    O    N    J    O    U    R
```

Pour écrire un mot (ou une phrase), on doit donc placer, derrière l'adresse du CALL, la liste des valeurs ASCII des caractères composant le mot. Nous l'avons vu, l'instruction (BASIC) CALL, place le nombre de paramètres (7 dans notre exemple) dans A, et remplit ainsi une zone de données pointée par le registre IX.



Analyse du programme

• Sauvegarder en mémoire, à l'adresse &8ØØØ, le nombre de paramètres donc de caractères à afficher.

Le nombre de paramètres est placé dans A par l'instruction CALL. Or le registre A est utilisé comme entrée de la routine d'affichage. Il est donc nécessaire de sauvegarder cette information en mémoire. Nous avons choisi l'adresse &8ØØØ. Le registre HL maintiendra cette adresse.

• Calculer l'adresse du premier paramètre, c'est-à-dire du premier caractère à afficher. On le calcule dans IX.

IX = IX + 2 ×(nombre de caractères − 1) (voir schéma).

• Tant qu'il y a un caractère à afficher (tant que le contenu de la mémoire &8ØØØ est non nul).
- Afficher le caractère (pointé par IX).
- Décrémenter deux fois le pointeur de paramètres (IX = IX − 2).
- Décrémenter le nombre de caractères à afficher (donc décrémenter l'octet mémoire d'adresse &8ØØØ).

Adresse	Instruction	Code	
&61ØØ	LD HL, &8ØØØ	**&21**	
		&ØØ	Valeur
		&8Ø	
&61Ø3	LD (HL), A	**&77**	
&61Ø4	DEC A	**&3D**	
&61Ø5	LD B, &ØØ	**&Ø6**	
		&ØØ	Valeur
&61Ø7	LD C, A	**&4F**	
&61Ø8	ADD IX, BC	**&DD**	
		&Ø9	
&61ØA	ADD IX, BC	**&DD**	
		&Ø9	
&61ØC	LD A, &ØØ	**&3E**	
(Boucle)		&ØØ	Valeur
&61ØE	CP (HL)	**&8E**	
&61ØF	JP Z, &6122	**&CA**	
		&22	Adresse
		&61	(Fin)
&6112	LD A, (IX + &ØØ)	**&DD**	
		&7E	
		&ØØ	Déplacement
&6115	PUSH HL	**&E5**	
&6116	CALL &BB5D	**&CD**	
		&5D	Adresse
		&BB	
&6119	POP HL	**&E1**	
&611A	DEC IX	**&DD**	
		&2B	
&611C	DEC IX	**&DD**	
		&2B	
&611E	DEC (HL)	**&35**	
&611F	JP &61ØC	**&C3**	
		&ØC	Adresse
		&61	(Boucle)
&6122	RET	**&C9**	
			(Fin)

Remarquez que nous avons sauvegardé HL lors de l'appel de la routine d'affichage. En effet, cette routine modifie les registres du jeu principal. En revanche il n'est pas nécessaire de sauvegarder IX.

Faites CALL &61ØØ, &42, &4F, &4E, &4A, &4F, &55, &52. BONJOUR est affiché.

Faites d'autres essais pour vérifier que vous pouvez afficher des mots ou phrases de longueurs différentes. En particulier, essayez simplement CALL &6100. Le programme que nous avons écrit vérifie, avant d'afficher un caractère, qu'il y en a bien un à afficher.

Conclusion

Vous le voyez, un programme équivalent à un simple PRINT nécessite une trentaine d'octets (et encore, en appelant une routine système !).

A vous de doser dans vos programmes le mélange BASIC-langage machine. La présentation du langage machine Z80, et celle des routines système de l'Amstrad qui va suivre, doivent vous permettre d'enchaîner des routines là où c'est absolument nécessaire. Nous ne vous conseillons pas de vous lancer dans d'énormes programmes machine (sauf si vous disposez d'un assembleur).

Maintenant, à vous de jouer. Apprendre le langage machine du Z80 vous permettra de mieux connaître votre Amstrad, et aussi d'apprécier l'extraordinaire puissance d'un langage évolué comme le BASIC.

Chapitre VII

Adresses Système

Les instructions RESTART dans l'Amstrad

Le Z8Ø présente huit instructions RESTART d'un seul octet de code machine, qui permettent d'appeler des programmes à une adresse bien déterminée. RST Ø, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6 et RST 7 appellent les programmes commençant aux adresses &ØØØØ, &ØØØ8, &ØØ1Ø, &ØØ18, &ØØ2Ø, &ØØ28, &ØØ3Ø et &ØØ38.

Rappelez-vous, ces adresses se situent dans la ZONE SYSTÈME. Nous avions dit que les 64 premiers octets de la ROM basse étaient recopiés dans la ZONE SYSTÈME RAM. Ainsi, quel que soit l'état de la sélection de la mémoire basse (soit la ROM basse, soit la RAM basse) une instruction RST provoque toujours le même traitement.

Les traitements effectués sur les instructions RST sont fondamentaux pour le fonctionnement de l'Amstrad. Ils se classent en deux familles.

Restarts "système"

RST Ø (&ØØØØ). L'initialisation

Par la force des choses (du Z8Ø) cette instruction, provoquant l'appel du traitement en &ØØØØ, réinitialise complètement l'Amstrad. En effet lors de la mise sous tension, le Z8Ø se branche automatiquement à l'adresse &ØØØØ. Le programmeur doit donc placer à cette adresse son programme d'initialisation. La (ré)initialisation a pour effet d'effacer complètement le programme BASIC en mémoire, d'annuler toutes les redéfinitions de touches, de caractères... Elle exécute aussi une réinitialisation des circuits d'entrée/sortie.

RST 7. (&ØØ38). Interruption

Nous l'avons vu, le Z8Ø de l'Amstrad est programmé en mode 1. Ainsi, toutes les interruptions provoquent l'exécution du programme commençant à l'adresse &ØØ38.

RST 6. (&ØØ3Ø). Utilisateur

L'instruction RST6 de code &F7 est laissée à l'utilisateur. Celui-ci peut donc placer aux adresses &ØØ3Ø jusqu'à &ØØ37 le début d'un programme. Une instruction RST est idéale pour réaliser les "points d'arrêt" dans un debugger. Nous vous proposons dans notre programme MONITEUR cette utilisation de RST6.

(Bien que le programme utilisateur placé sur RST6 ne puisse être recopié en ROM, il est possible quand même d'effectuer à tout instant un RST6, donc même si la ROM basse est sélectionnée. Le programme RST6 placé en ROM se charge de sélectionner la RAM basse et de faire exécuter le programme utilisateur en RAM.)

La navigation dans la mémoire

Les autres instructions de RESTART sont utilisées dans l'Amstrad pour la navigation dans la mémoire. Celle-ci, a priori, n'est pas si simple que cela, puisque le microprocesseur a le choix entre 64 k-octets de RAM et de nombreux boîtiers de ROM. Les instructions RESTART permettent d'appeler un programme, non seulement en précisant son adresse, mais aussi le boîtier (ROM ou RAM)

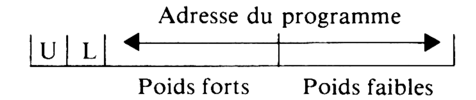
où il se trouve. Bien sûr, en fin d'exécution du programme appelé, l'état initial de la sélection des boîtiers est restitué, ce qui évite bien des soucis au programmeur !

RST 1. (&ØØØ8). Saut à un programme en mémoire basse

Cette instruction est utilisée dans l'Amstrad pour sauter à un programme placé en mémoire basse. Pour cela on doit faire suivre le code machine de RST1 **(&CF)** de deux octets précisant l'adresse de la routine en mémoire basse où sauter, et l'état voulu des boîtiers. Le format est le suivant :

&CF, poids faibles "adresse", poids forts "adresse" + sélection.

Les deux octets doivent être ainsi constitués.



Les quatorze bits de droite indiquent l'adresse de la routine. Cette adresse est donc comprise entre &ØØØØ et &3FFF, ce qui correspond bien à la mémoire basse.

Le bit L (Low) indique à 1 que le programme appelé se trouve en RAM basse, à Ø qu'il se trouve en ROM basse.

Le bit U (Upper) permet de plus au programme de valider (avec U = 1) la RAM haute, ou la ROM haute (avec U = Ø).

Exemples
&CF &3Ø &Ø8 est un RST1 avec Ø Ø &Ø83Ø

Cette instruction a pour effet de sauter au programme d'adresse &83Ø se trouvant dans la ROM basse. La ROM haute, sélectionnée à cet instant, est validée.

&CF &3Ø &C8 est un RST1 avec 1 1 &Ø83Ø

Cette instruction a pour effet de sauter au programme d'adresse &83Ø se trouvant dans la RAM basse. La ROM haute est invalidée.

RST 2. (&ØØ1Ø). Appel voisin

Cette instruction peut être utilisée dans des programmes en extension, occupant jusqu'à quatre ROM voisines. Elle permet d'appeler,

à partir d'une des quatre ROM, un programme dans une de ces ROM sans préciser la sélection des boîtiers. Cette sélection est effectuée à partir de l'adresse "64 K" fournie. L'utilisateur ne peut pas s'en servir.

RST 3. (&ØØ18). Appel en tout endroit

Cette instruction permet, dans l'Amstrad, d'appeler un programme n'importe où en RAM, ou dans n'importe quel boîtier ROM. Pour cela, on doit faire suivre l'instruction RST3 (de code machine **&DF**) de deux octets donnant l'adresse de "l'adresse lointaine". Le format de l'"adresse lointaine" est le suivant :
Octet 1 : poids faibles de l'adresse du programme à appeler.
Octet 2 : poids forts de l'adresse du programme à appeler.
Octet 3 : validation - sélection des ROM.

Ce dernier octet a la signification suivante :
— de &ØØ à &FB : la ROM haute du numéro donné (Ø à 251) est sélectionnée et validée. La ROM basse est invalidée.
— de &FC à &FF : la sélection de la ROM haute n'est pas modifiée, mais l'état validé-invalidé des ROM basse et haute est ainsi modifié.

- &FC ROM haute validée, ROM basse validée.
- &FD ROM haute validée, ROM basse invalidée.
- &FE ROM haute invalidée, ROM basse validée.
- &FF ROM haute invalidée, ROM basse invalidée.

Exemple

&DF &3Ø &Ø8 appelle le programme dont l'"adresse lointaine" est placée à l'adresse &83Ø.

Si l'on a placé à cette adresse &Ø2, &D8, &4Ø, on appelle le programme placé à l'adresse &D8Ø2 dans la ROM &4Ø (64), et la ROM basse est invalidée.

Si l'on a placé à cette adresse &Ø2 &28 &FF, on appelle le programme placé en RAM à l'adresse &28Ø2. Les ROM haute et basse sont invalidées.

Vous remarquez que cette instruction peut servir uniquement à valider une ROM durant le temps d'exécution d'un programme en RAM.

Ainsi, si on utilise toujours l'adresse &83Ø pour placer l'adresse lointaine et si on y place &ØØ, &6Ø, &FC, un RST3 &83Ø fait exécuter le

programme placé en &6000 (donc en RAM), avec ROM haute et basse validées.
Dans notre MONITEUR, nous utilisons ainsi le RST3 pour pouvoir lire ("dumper") les ROM, auxquelles on n'a pas "normalement" accès dans un programme BASIC. (L'instruction PEEK va toujours lire la RAM, elle utilise d'ailleurs probablement le RESTART suivant.)

RST 4 (&0020). Lecture de la RAM

Cette instruction de code **&E7** est équivalente dans l'Amstrad à l'instruction LD A, (HL), excepté qu'elle charge toujours dans l'accumulateur le contenu de la RAM pointée par HL, quel que soit l'état de validation des ROM.

Exemple
LD HL, &3000
RST 4

place dans A le contenu de la RAM d'adresse &3000, même si à ce moment la ROM basse est validée.

RST 5. (&0028). Saut vers le firmware

Cette instruction est similaire à RST 1. Elle permet de sauter à un programme se situant soit dans la RAM moyenne, soit dans la ROM basse. La ROM haute reste dans l'état où elle était. La ROM basse est validée.

Cette instruction doit son nom au fait qu'elle est extrêmement pratique pour sauter à un programme de la ROM système (firmware).

Le code de l'instruction **(&EF)** doit être suivi de l'adresse du programme.

Exemples
&EF &30 &08 saute au programme d'adresse &830 de la ROM basse.

&EF &30 &58 saute au programme d'adresse &5830 de la RAM moyenne. La ROM basse est validée.

Remarques

— Les instructions RST1 et RST5 sont équivalentes à des **sauts** et non à des appels. Pour **appeler** une routine système, il suffit d'appe-

ler le programme effectuant le saut (RST1 ou RST5) à la routine système voulue.
— Les adresses de routines système que nous allons maintenant vous donner ne sont pas réellement les adresses en ROM des routines système. Ce sont les adresses dans la zone RAM, BLOC DE SAUTS, des instructions de sauts aux routines système. Ces instructions de sauts utilisent les RESTART 1 et 5.

Vous n'avez donc pas à vous soucier, pour utiliser les routines système, de l'état de la sélection de la ROM et de la RAM. Il suffit de sauter à (JP), ou d'appeler (CALL) les adresses que nous indiquons.

Quelques sous-programmes

La liste des sous-programmes machine qui suit vous permettra d'augmenter vos possibilités de programmation. En effet, la plupart des sous-programmes présentés ne trouvent pas leurs équivalents dans la panoplie des instructions BASIC. Nous avons cependant indiqué quelques sous-programmes équivalents à une instruction BASIC, mais dont la rapidité d'exécution peut être "critique". Nous avons précisé dans ces cas l'instruction BASIC correspondante.

Action : Lecture ou attente d'une touche en provenance du clavier.

Adresse : &BB18

Entrées : —

Sorties : Le code de la touche est dans l'accumulateur A.
Le "Carry Flag" est à 1.

Action : Lecture d'une touche en provenance du clavier.

Adresse : &BB1B

Équivalent : INKEY$

Entrées : —

Sorties :	Si une touche est appuyée. Alors - le code de la touche est dans l'accumulateur A, - le "Carry Flag" est à 1. Sinon - l'accumulateur A est détruit, - le "Carry Flag" est à Ø.

Action :	Autorisation de recevoir un "break".
Adresse :	&BB45
Entrées :	L'adresse du sous-programme de "break" se trouve dans le registre DE Le registre C contient l'adresse de sélection de la mémoire morte.
Sortie :	Les registres AF, BC, DE, HL sont détruits.

Action :	Interdiction de recevoir un "break".
Adresse :	&BB48
Entrées :	—
Sorties :	Les registres AF et HL sont détruits.

Action :	Simulation d'un "break" puis interdiction de recevoir un "break" ultérieur.
Adresse :	&BB4B
Entrées :	—
Sorties :	Les registres AF et HL sont détruits.

Action :	Initialisation totale de l'écran texte.
Adresse :	&BB4E
Entrées :	—
Sorties :	Les registres AF, BC, DE et HL sont détruits.

Action :	Autorisation/Interdiction d'écrire des caractères sur le canal en cours.
Adresse :	&BB54/&BB57
Entrées :	—
Sorties :	Le registre AF est détruit.

Action : Écrire un caractère sur l'écran et exécuter la fonction correspondante au code si la valeur du caractère est inférieure à 32.

Adresse : &BB5A

Équivalent : PRINT à l'écran.

Entrées : Le caractère à afficher se trouve dans l'accumulateur A.

Sorties : —

Action : Écrire un caractère sur l'écran.

Adresse : &BB5D

Équivalent : PRINT à l'écran sous mode graphique.

Entrées : Le caractère à afficher se trouve dans l'accumulateur A.

Sorties : Les registres AF, BC, DE et HL sont détruits.

Action : Positionnement du curseur dans la fenêtre en cours.

Adresse : &BB75

Équivalent : LOCATE

Entrées : Le numéro de colonne est dans le registre H.
Le numéro de ligne est dans le registre L.

Sorties : Les registres AF et HL sont détruits.

Action : Inversion vidéo en mode texte sur le canal en cours.

Adresse : &BB9C

Entrées : —

Sorties : Les registres AF et HL sont détruits.

Action : Initialisation totale de l'écran graphique.

Adresse : &BBBA

Entrées : —

Sorties : Les registres AF, BC, DE et HL sont détruits.

Action : Positionnement absolu/relatif du curseur graphique.

Adresse : &BBC0/&BBC3

Équivalents : MOVE/MOVER

Entrées : Le registre DE contient l'abscisse absolue/relative du curseur.
Le registre HL contient l'ordonnée absolue/relative du curseur.

Sorties : Les registres AF, BC, DE et HL sont détruits.

Action : Effacement de l'écran. L'écran prend la couleur de l'encrier zéro.

Adresse : &BC14

Entrées : —

Sorties : Les registres AF, BC, DE et HL sont détruits.

Action : Colorie un rectangle sur l'écran.

Adresse : &BC44

Entrées : Le registre A contient la valeur de l'encre à utiliser.
Les registres H, D, L et H contiennent les informations sur la taille et la position du rectangle selon le principe utilisé par l'instruction WINDOW.

Sorties : Les registres AF, BC, DE et HL sont détruits.

Action : Effacement de la fenêtre graphique. La fenêtre prend la couleur du papier.

Adresse : &BBDB

Entrées : —

Sorties : Les registres AF, BC, DE et HL sont détruits.

Action : Allumage d'un point absolu/relatif de l'écran graphique.

Adresse : &BBEA/&BBED

Équivalents : PLOT/PLOTR

Entrées : Le registre DE contient l'abscisse absolue/relative du point.
Le registre HL contient l'ordonnée absolue/relative du point.

Sorties : Les registres AF, BC, DE et HL sont détruits.

Action : Tracé d'une droite en coordonnées absolues/relatives.

Adresse : &BBF6/&BBF9

Équivalents : DRAW/DRAWR

Entrées : L'abscisse absolue/relative de l'extrémité de la droite est dans le registre DE.
L'ordonnée absolue/relative de l'extrémité de la droite est dans le registre HL.

Sorties : Les registres AF, BC, DE et HL sont détruits.

Moniteur avec points d'arrêt

En conclusion de ce chapitre, nous vous proposons ce moniteur qui vous permettra de faire la mise au point de vos programmes écrits en langage machine. Il offre les fonctions suivantes :

— dump ou visualisation de la mémoire RAM et ROM. Vous frappez D, puis vous indiquez les adresses de début et de fin de la zone que vous voulez visualiser. Le moniteur affiche, par lignes de huit octets, le contenu de la mémoire.

— choix de la ROM ou de la RAM pour le dump. Vous frappez C, puis N (pour Non) si la mémoire actuellement sélectionnée ne vous satisfait pas ;

— modification de la mémoire. Vous frappez M, vous indiquez l'adresse de l'octet que vous désirez modifier, puis la nouvelle valeur de l'octet. Le programme vous propose alors de modifier l'octet suivant, et ainsi de suite jusqu'à ce que vous frappiez X (pour sortir de la fonction) ;

— lancement d'un programme, avec ou sans point d'arrêt, à partir ou non d'un point d'arrêt. L'utilisation la plus simple de cette fonction consiste à lancer un programme en indiquant uniquement son adresse de début, sans indiquer l'adresse de fin. Le moniteur lance alors ce programme jusqu'à son exécution finale. Il est aussi possible d'indiquer l'"adresse de fin" d'un point d'arrêt. Celui-ci doit obligatoirement se trouver en début d'une instruction machine, afin de

pouvoir fonctionner correctement. Le programme est exécuté jusqu'à cette instruction ; il est ensuite possible de le relancer à partir de cette instruction (en indiquant ou pas un nouveau point d'arrêt). L'état des registres du Z80 est sauvegardé lors de l'exécution du point d'arrêt, puis restitué lors du redémarrage à partir de ce point, ce qui rend possible l'utilisation d'un point d'arrêt en tout endroit de votre programme ;

— visualisation du contenu des registres après un point d'arrêt. Vous frappez R, l'état des registres AF, BC, DE, HL, IX, IY, PC et SP est affiché. C'est l'état des registres du Z80 au moment du dernier point d'arrêt que vous avez effectué.

Voici un exemple d'utilisation du moniteur : vous voulez écrire le programme suivant à l'adresse &8500.

```
&8500   &11   &00   &02      LD   DE, &200 abscisse
&8503   &21   &00   &01      LD   HL, &100 ordonnée
&8506   &CD   &F6   &BB      CALL &BBF6   tracé de droite
&8509   &C9                  RET
```

La fonction M vous permet d'entrer le code de votre programme. La fonction D permet de vérifier que le contenu de la mémoire située entre &8500 et &8509 est correct.

Vous pouvez alors lancer le programme en frappant L, en indiquant l'adresse de début — 8500 —, et en n'indiquant pas l'adresse de fin. Le programme est entièrement exécuté ; la droite est tracée.

Vous pouvez aussi, pour vérifier que vos instructions sont correctes, exécuter ce programme avec des points d'arrêt. Pour cela, vous lancez le programme à partir de l'adresse de début — 8500 — jusqu'à l'adresse de fin — 8503 —, début de la seconde instruction. Vérifiez ensuite, en frappant R, que le contenu de DE est bien &200. Relancez le programme à partir du point d'arrêt en frappant L et en n'indiquant pas d'adresse de début (vous "conservez" l'adresse de début proposée — 8503 — en frappant ENTER directement lorsque le programme vous demande l'adresse de début). Indiquez l'adresse du point d'arrêt — 8506 — (début de la troisième instruction) lorsque le moniteur vous demande l'adresse de fin. Vous pouvez vérifier ensuite que le contenu de HL est bien &100 en frappant R.

Relancez le programme pour exécution complète en frappant L et en

n'indiquant ni l'adresse de début, ni l'adresse de fin (frappez deux fois ENTER). La droite est tracée.

Toutes les valeurs doivent être fournies en hexadécimal, sans le caractère "&" ; les lettres doivent être frappées en minuscules.

```
1 '===================================
2 '= moniteur pour programmes machine =
3 '===================================
4 '
5 GOSUB 1000                      'initialisation
7 '
8 ' boucle principale
9 '------------------
10 com$=INKEY$                                        'lecture commande
20 IF com$="m" THEN GOSUB 2000: GOSUB 10000 'modification memoire
30 IF com$="d" THEN GOSUB 3000: GOSUB 10000 'dump memoire
40 IF com$="r" THEN GOSUB 4000: GOSUB 10000 'valeur registres
50 IF com$="l" THEN GOSUB 5000: GOSUB 10000 'lancement programme
60 IF com$="c" THEN GOSUB 6000: GOSUB 10000 'choix memoire
70 IF com$<>"x" THEN GOTO 10
98 '
99 '  sortie du debugger
100 '-------------------
110 END
997 '
998 ' initialisation
999 '----------------
1000 MODE 1:INK 0,1:INK 1,24:INK 2,0:INK 3,6 'mode et couleurs
1005 WINDOW #0,1,40,2,22                      'fenetre centrale
1010 WINDOW #1,1,40,23,25                     'fenetre menu
1015 WINDOW #7,1,40,1,1                       'fenetre presentation
1020 BORDER 9
1025 PAPER #0,0:PEN #0,1:CLS #0               'caracteres 0
1030 PAPER #1,2:PEN #1,3:CLS #1               'caracteres 1
1035 PAPER #7,2:PEN #7,3:CLS #7               'caracteres 7
1040 PRINT #7,"moniteur          memoire de 8000 a 9c00"
1045 MEMORY &8000                'reservation memoire machine
1050 mem=0                       'selection ram pour dump
1088 '
1089 ' programme machine en data
1090 DATA &a0,&9c,&c3,&60,&9c
1100 DATA &00,&00,&00,&00,&00,&00,&00,&00
1110 DATA &00,&00,&00,&00,&00,&80,&fe,&9f
1120 DATA &00,&00,&00,&00,&d0,&9c,&fc,&00
1130 DATA &cb,&4f,&ca,&14,&9c,&dd,&66,&03
1140 DATA &dd,&6e,&02,&22,&8c,&9e,&21,&fe
1150 DATA &9f,&22,&8e,&9e,&cb,&47,&ca,&25
1160 DATA &9c,&dd,&66,&01,&dd,&6e,&00,&7e
1170 DATA &32,&92,&9e,&36,&f7,&ed,&73,&90
1180 DATA &9e,&ed,&7b,&8e,&9e,&2a,&8c,&9e
1190 DATA &e5,&2a,&8a,&9e,&e5,&f1,&ed,&4b
1200 DATA &80,&9e,&ed,&5b,&82,&9e,&2a,&84
1210 DATA &9e,&dd,&2a,&86,&9e,&fd,&2a,&88
1220 DATA &9e,&c9,&00,&00,&00,&00,&00,&00
1230 DATA &00,&00,&00,&00,&00,&00,&00,&00
1240 DATA &00,&00,&00,&00,&00,&00,&00,&00
1250 DATA &ed,&43,&80,&9e,&ed,&53,&82,&9e
1260 DATA &22,&84,&9e,&dd,&22,&86,&9e,&fd
1270 DATA &22,&88,&9e,&f5,&e1,&22,&8a,&9e
1280 DATA &e1,&2b,&22,&8c,&9e,&ed,&73,&8e
1290 DATA &9e,&3a,&92,&9e,&77,&c3,&a0,&9c
```

```
1300 DATA &00,&00,&00,&00,&00,&00,&00,&00
1310 DATA &00,&00,&00,&00,&00,&00,&00,&00
1320 DATA &00,&00,&00,&00,&00,&00,&00,&00
1330 DATA &ed,&7b,&90,&9e,&c9,&00,&00,&00
1340 DATA &00,&00,&00,&00,&00,&00,&00,&00
1350 DATA &00,&00,&00,&00,&00,&00,&00,&00
1360 DATA &00,&00,&00,&00,&00,&00,&00,&00
1370 DATA &df,&94,&9e,&c9,&00,&00,&00,&00
1380 DATA &00,&00,&00,&00,&00,&00,&00,&00
1390 DATA &dd,&6e,&00,&dd,&66,&01,&7e,&32
1400 DATA &98,&9e,&c9,&00,&00,&00,&00,&00
1408 '
1409 'adresse retour basic sommet pile machine
1410 adresse=&9FFE: adresse=adresse+2^16
1420 FOR i=1 TO 2: GOSUB 1900:NEXT
1429 'instruction en &30 (saut de RST 6)
1430 adresse=&30
1440 FOR i=1 TO 3: GOSUB 1900:NEXT
1449 'donnees et adresse lointaine
1450 adresse=&9E80: adresse=adresse+2^16
1460 FOR i=1 TO 24: GOSUB 1900:NEXT
1469 'programmes
1470 adresse=&9C00: adresse=adresse+2^16
1480 FOR i=1 TO 224: GOSUB 1900:NEXT
1499 '
1500 GOSUB 10000            'affiche menu general
1510 RETURN
1888 '
1889 ' sous-programme entree programme machine
1900 READ a:POKE adresse,a:adresse=adresse+1:RETURN
1996 '
1997 '======================
1998 ' modification memoire
1999 '======================
2000 GOSUB 10100                            'affiche menu
2010 INPUT "adresse";a$: IF a$="x" THEN RETURN
2020 GOSUB 11000: IF erreur=1 THEN GOTO 2010
2030 adresse=a-1
2038 '
2039 ' boucle d'entree des valeurs a partir de l'adresse
2100 adresse=adresse+1
2105 PRINT HEX$(adresse)" :";              'affiche adresse
2110 a=PEEK(adresse): PRINT HEX$(a)" ,";   'ancien contenu
2120 INPUT"ok";a$: IF a$="x" THEN RETURN
2130 IF a$="" THEN GOTO 2100               'pas de modification
2140 GOSUB 11000:IF erreur=1 OR a>&FF
               THEN GOTO 2120              'conversion et controle
2150 POKE adresse,a                        'prise en compte
2160 GOTO 2100                             'adresse suivante
2996 '
2997 '======================
2998 ' dump memoire
2999 '======================
3000 GOSUB 10200
3010 INPUT "debut";a$: IF a$="x" THEN RETURN
3020 GOSUB 11000: IF erreur=1 THEN GOTO 3010 ELSE deb=a
3030 INPUT "fin  ";a$: IF a$="x" THEN RETURN
3040 GOSUB 11000: IF erreur=1 THEN GOTO 3030 ELSE fin=a
3050 IF fin<deb THEN GOTO 3010
3098 '
3099 ' affichage des octets memoire d'adresses deb a fin
3100 PRINT HEX$(deb,4)"  ";                    'affiche 1ere adresse
3110 deb=deb-1: long=fin-deb
3120 FOR i=1 TO long
3125      deb=deb+1
```

```
3130      IF deb-8*INT(deb/8)=0 AND i<>1
          THEN PRINT: PRINT HEX$(deb,4)"   "; 'affiche adresses =*8
3140      IF mem=0
          THEN cnt=PEEK(deb): GOTO 3160       'peek en ram
3150      CALL &9CC0,deb: cnt=PEEK(&9E98)     'lecture rom
3160      PRINT HEX$(cnt,2)" ";               'affiche contenu
3170 NEXT
3800 RETURN
3996 '
3997 '======================
3998 ' etat registres
3999 '======================
4000 PRINT "AF=";: adresse=&9E8A: GOSUB 4900
4010 PRINT "BC=";: adresse=&9E80: GOSUB 4900
4020 PRINT "DE=";: adresse=&9E82: GOSUB 4900
4030 PRINT "HL=";: adresse=&9E84: GOSUB 4900
4035 PRINT
4040 PRINT "IX=";: adresse=&9E86: GOSUB 4900
4050 PRINT "IY=";: adresse=&9E88: GOSUB 4900
4060 PRINT "SP=";: adresse=&9E8E: GOSUB 4900
4070 PRINT "PC=";: adresse=&9E8C: GOSUB 4900
4080 RETURN
4898 '
4899 ' affiche contenu de l'adresse (contenant registre)
4900 adresse=adresse+2^16
4910 a=PEEK(adresse+1): PRINT HEX$(a,2);
4920 a=PEEK(adresse): PRINT HEX$(a,2)" ";
4930 RETURN
4996 '
4997 '======================
4998 ' lancement programme
4999 '======================
5000 GOSUB 10400
5005 PRINT "ad.debut ";:adresse=&9E8C:GOSUB 4900 'affiche ad.debut
5010 INPUT "ok";a$: IF a$="x" THEN RETURN
5020 IF a$="" THEN nbp=0: GOTO 5050          'ad.deb pas modifiee
5030 GOSUB 11000: IF erreur=1 THEN GOTO 5010
5040 nbp=2: deb=a                            'ad.deb modifiee
5050 INPUT "ad.fin          ";a$: IF a$="x" THEN RETURN
5060 IF a$="" THEN GOTO 5100                 'pas d'ad.fin
5070 GOSUB 11000: IF erreur=1 THEN GOTO 5050
5080 nbp=nbp+1: fin=a                        'point d'arret
5090 '
5099 ' lancement du programme
5100 IF nbp=0 THEN CALL &9C00
5110 IF nbp=1 THEN CALL &9C00,fin
5120 IF nbp=2 THEN CALL &9C00,deb,0
5130 IF nbp=3 THEN CALL &9C00,0,deb,fin
5140 RETURN
5996 '
5997 '======================
5998 ' choix memoire
5999 '======================
6000 GOSUB 10500
6010 IF mem=0 THEN PRINT "ram"; ELSE PRINT "rom";
6020 INPUT " ";a$: IF a$<>"n" THEN RETURN
6030 mem=mem+1: IF mem=2 THEN mem=0
6040 GOTO 6010
8999 '
9000 '======================
9001 ' sous-programmes
9002 '======================
9989 '----------------------
9990 ' affichage des menus
9991 '----------------------
```

```
9992 '
9998 ' menu general
9999 '----------------
10000 CLS #1: PRINT
10010 PRINT #1,"C choix    D dump    L lancement"
10020 PRINT #1,"M modification      R registres"
10030 PRINT #1,"X fin"
10040 RETURN
10097 '
10098 ' menu modification
10099 '------------------
10100 CLS #1
10110 PRINT #1,"indiquer adresse puis valeurs"
10120 PRINT #1,"X fin";
10130 RETURN
10197 '
10198 ' menu dump
10199 '--------------
10200 CLS #1
10210 PRINT #1,"memoire actuelle ";
10220 IF mem=0 THEN PRINT #1,"RAM" ELSE PRINT #1,"ROM"
10230 PRINT #1,"indiquer adresses"
10240 PRINT #1,"X fin"
10250 RETURN
10397 '
10398 ' menu lancement
10399 '------------------
10400 CLS #1
10410 PRINT #1,"autre adresse debut ou enter"
10420 PRINT #1,"adresse fin (point arret) ou enter"
10430 PRINT #1,"X fin"
10440 RETURN
10497 '
10498 ' menu choix
10499 '--------------
10500 CLS #1
10510 PRINT #1,"frapper N pour modifier le choix"
10520 PRINT #1,"une autre touche  fin";
10530 RETURN
10994 '
10995 '
10996 '----------------------------------------
10997 ' conversion chaine de caractere en valeur
10998 ' entree a$: sortie a ou erreur = 1
10999 '----------------------------------------
11000 erreur = 0: lg=LEN(a$)                        'controle longueur
11010 IF lg>4 OR lg=0 THEN erreur = 1: RETURN
11020 FOR i=1 TO lg                                 'controle que les
11030     ca$=MID$(a$,i,i+1): ca=ASC(ca$)           'caracteres
11040     IF ca<48 OR ca>102 THEN erreur = 1   'sont entre 0 et 9
11050     IF ca>57 AND ca<97 THEN erreur = 1   'ou a et f
11060 NEXT                                          'puis convertit
11070 IF erreur = 0 THEN a=VAL("&"+a$): IF a<0 THEN a=a+2^16
11080 RETURN
49988 '
49989 '======================
50000 ' programmes machines
50001 '======================
50004 '
50005 ' la memoire de 9c00 ---> 9fff est reservee au moniteur
50010 '
50020 ' pile du moniteur
50021 '===================
50022 '
50030 '     elle est placee de 9f00 a 9fff
```

```
50040 '    le sommet de la pile contient l'adresse du programme
50050 '    effectuant le retour d'un programme machine vers le
50060 '    BASIC, ce programme est place en 9ca0
50070 '    en 9fff ---> 9c, en fffe ---> a0
50090 '
50100 ' donnees moniteur
50101 '====================
50102 '
50110 '   1. pour point d'arret
50115 '   -------------------------
50120 ' 9e80 sauvegarde BC                          * "contexte"
50130 ' 9e82 sauvegarde DE                          * "machine"
50140 ' 9e84 sauvegarde HL                          *
50150 ' 9e86 sauvegarde IX                          *
50160 ' 9e88 sauvegarde IY                          *
50170 ' 9e8a sauvegarde AF                          *
50180 ' 9e8c sauvegarde PC (adresse debut)          *
50190 ' 9e8e sauvegarde SP (pile debugger)          *
50195 '
50200 ' 9e90 sauvegarde SP du BASIC  -
50210 ' 9e92 sauvegarde octet ecrase par RST 6
50215 '
50220 '   2. pour lecture ROM
50225 '   ---------------------
50230 ' 9e94 3 octets de l'adresse lointaine appelee sur RST 3
50235 ' le programme est situe en 9cd0, les ROMs sont validees
50240 ' en 9e94 ---> d0, 9e95 ---> 9c, 9e96 ---> fc
50250 ' 9e98 resultat de la lecture de la ROM
50270 '
50300 ' programmes
50301 '===============
50302 '
50310 '   1. saut sur RST 6
50311 '   -------------------
50320 ' 0030  c3 60 9c       jp    9c60       saut au traitement
50330 '
50340 '   2. appel basic pour lancement programme
50341 '   -----------------------------------------
50350 ' 9c00  cb 4f          bit   1,a        nbp 2 ou 3 ?
50360 ' 9c02  ca 14 9c       jp    z,9c14     <2, pas ad.deb
50370 ' 9c05  dd 66 03       ld    h,(ix+3)   p.forts ad.deb
50380 ' 9c08  dd 6e 02       ld    l,(ix+2)   p.faibs ad.deb
50390 ' 9c0b  22 8c 9e       ld    (9e8c),hl  memorisee
50400 ' 9c0e  21 fe 9f       ld    hl,9ffe    valeur init pile
50410 ' 9c11  22 8e 9e       ld    (9e8e),hl  memorisee
50415 '
50420 ' 9c14  cb 47          bit   0,a        nbp 1 ou 3 ?
50430 ' 9c16  ca 25 9c       jp    z,9c25     <>1 ou 3 pas ad.fin
50440 ' 9c19  dd 66 01       ld    h,(ix+1)   p.forts ad.fin
50450 ' 9c1c  dd 6e 00       ld    l,(ix+0)   p.faibs ad.fin
50460 ' 9c1f  7e             ld    a,(hl)     ancien code
50470 ' 9c20  32 92 9e       ld    (9e92),a   memorise
50480 ' 9c23  36 f7          ld    (hl),f7    remplace par RST6
50485 '
50490 ' 9c25  ed 73 90 9e    ld    (9e90),sp  sauve pile basic
50500 ' 9c29  ed 7b 8e 9e    ld    sp,(9e8e)  pile machine
50510 ' 9c2d  2a 8c 9e       ld    hl,(9e8c)  adresse debut
50520 ' 9c30  e5             push  hl         empilee pour RET
50530 ' 9c31  2a 8a 9e       ld    hl,(9e8a)  restitution AF
50540 ' 9c34  e5             push  hl         par pile
50550 ' 9c35  f1             pop   af         via HL
50560 ' 9c36  ed 4b 80 9e    ld    bc,(9e80)  restitution
50570 ' 9c3a  ed 5b 82 9e    ld    de,(9e82)  autres
50580 ' 9c3e  2a 84 9e       ld    hl,(9e84)  registres
50590 ' 9c41  dd 2a 86 9e    ld    ix,(9e86)
```

```
50600 ' 9c45  fd 2a 88 9e    ld    iy,(9e88)
50610 ' 9c49  c9             ret                 ---> execution
50630 '
50640 '    3. retour machine sur RST6
50641 '    --------------------------
50650 ' 9c60  ed 43 80 9e    ld    (9e80),bc    sauvegarde
50660 ' 9c64  ed 53 82 9e    ld    (9e82),de    registres
50670 ' 9c68  22 84 9e       ld    (9e84),hl
50680 ' 9c6b  dd 22 86 9e    ld    (9e86),ix
50690 ' 9c6f  fd 22 88 9e    ld    (9e88),iy
50700 ' 9c73  f5             push  af           sauvegarde AF
50710 ' 9c74  e1             pop   hl           par pile
50720 ' 9c75  22 8a 9e       ld    (9e8a),hl    via HL
50730 ' 9c78  e1             pop   hl           depile ad.ret RST6
50740 ' 9c79  2b             dec   hl           ad.fin
50750 ' 9c7a  22 8c 9e       ld    (9e8c),hl    memorisee---:ad.deb
50760 ' 9c7d  ed 73 8e 9e    ld    (9e8e),sp    sauvegarde pile
50770 ' 9c81  3a 9e 92       ld    a,(9e92)     ancien code
50780 ' 9c84  77             ld    (hl),a       restitue
50790 ' 9c85  c3 a0 9c       jp    9ca0         saute retour basic
50810 '
50820 '    4. retour machine vers basic
50821 '    ----------------------------
50830 ' 9ca0  ed 7b 90 9e    ld    sp,(9e90)    restitue pile basic
50840 ' 9ca4  c9             ret                retour basic
50860 '
50870 '    5. appel pour lecture ROM
50880 '    -------------------------
50890 ' 9cc0  df 94 90       RST3  (9094)       appel lointain 9094
50900 ' 9cc3  c9             ret
50920 '
50930 '    6. lecture ROM
50931 '    --------------
50940 ' 9cd0  dd 66 01       ld    h,(ix+1)     p.forts ad a lire
50950 ' 9cd3  dd 6e 00       ld    l,(ix+0)     p.faibs ad a lire
50960 ' 9cd6  7e             ld    a,(hl)       lecture
50970 ' 9cd7  32 98 9e       ld    (9e98),a     memorisee pour basic
50980 ' 9cda  c9             ret
```

Commentaires

La partie écrite en langage machine est placée en mémoire lors de l'initialisation du programme (à partir de la ligne 1ØØØ). Nous avons indiqué en commentaire, à partir de la ligne 5ØØØØ, la signification du code machine.

Lors d'un lancement de programme, on appelle le programme placé en &9CØØ avec Ø, 1, 2 ou 3 paramètres suivant que l'utilisateur a indiqué ou non une adresse de fin et de début. Dans ces cas, l'adresse de fin est le dernier paramètre, l'adresse de début est l'avant-dernier paramètre de l'instruction BASIC CALL. Le programme placé en &9CØØ (n° 2) mémorise l'adresse de début, si celle-ci est indiquée, remplace l'octet de l'éventuel point d'arrêt par l'instruction RST 6

(code &F7). Il effectue ensuite un "changement de contexte" BASIC vers machine (changement de pile et registres). Il lance ensuite le programme.

Si un point d'arrêt a été placé, son exécution provoque l'appel du programme placé à l'adresse &3Ø, où nous avons placé l'instruction de saut à l'adresse &9C6Ø. Ce programme effectue un changement de contexte machine vers BASIC en sauvegardant aux adresses &9E8Ø à &9E8F le contenu des registres à ce moment. Il place dans l'octet du point d'arrêt son ancien contenu.

S'il n'y a pas de point d'arrêt, le programme se termine par l'exécution du programme placé en &9CAØ dont l'adresse a été placée au sommet de la pile.

Cette manière de procéder est complexe ; mais elle est indispensable pour assurer la fonction "point d'arrêt" sans perdre l'état des registres à chaque arrêt

Les programmes machine 5 et 6 assurent la lecture de la ROM. Le programme placé en &9CCØ est appelé avec le paramètre "adresse à lire". Il appelle le programme placé en &9CDØ par l'intermédiaire d'un RST3 validant les ROM. (Adresse lointaine placée en &9E94.) Ce programme place dans la mémoire &9E98 le résultat de la lecture.

La partie du programme écrit en BASIC ne présente pas de difficulté particulière, si ce n'est les appels de programmes machine avec paramètres (lignes 315Ø, 51ØØ à 513Ø). L'instruction MEMORY réserve de la mémoire placée au-dessus de &8ØØØ à l'utilisateur. Celui-ci ne doit pas utiliser la mémoire placée au-dessus de &9CØØ, cette zone étant réservée au moniteur.

Variations

Vous pouvez compléter ce moniteur par des programmes écrits uniquement en BASIC. Voici quelques fonctions que vous pouvez ajouter :

— modification des registres pour un point d'arrêt. Pour cela, il suffit de modifier les données placées de &9E8Ø à &9E8F ;

— dump de la mémoire sous forme de caractères. Il faut indiquer le contenu de la mémoire par CHR$(). Cette fonction permet d'interpréter plus facilement le contenu de la mémoire, en particulier là où est stocké le programme BASIC ;

— sortie sur imprimante du dump ;

— sauvegarde sur cassette ou disquette de vos programmes machine, et chargement ;

— éventuellement, si vous êtes ambitieux, un désassembleur qui vous permettra de lire le contenu de la mémoire directement en langage assembleur ;

— etc. !

Chapitre VIII

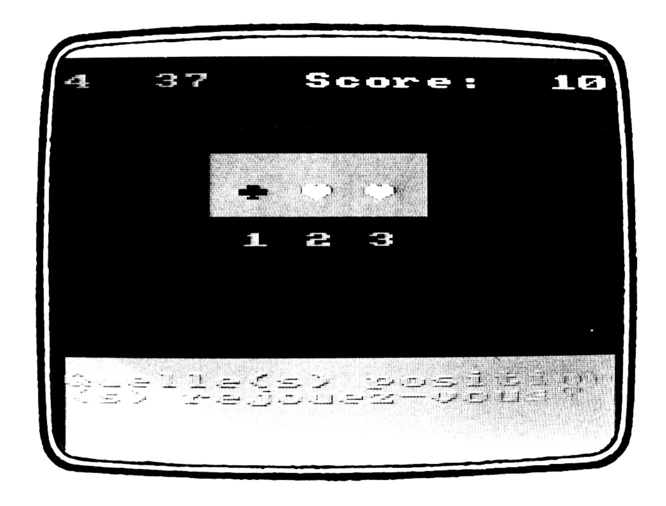
Quelques programmes

Les programmes qui suivent, ainsi que ceux proposés à la fin des chapitres précédents, sont destinés à illustrer les possibilités offertes par l'Amstrad. Un utilisateur non rompu au langage BASIC pourra s'en servir sans y apporter de modification. Ces programmes, tous inédits, constitueront une partie de sa "logithèque".

Mais les mêmes programmes peuvent servir de point de départ à des logiciels plus étoffés. Quelques notes, regroupées sous les paragraphes "variations", permettront aux passionnés d'étendre ou de modifier les fonctions proposées.

Nous avons essayé de structurer nos programmes pour faciliter leur lecture et leur portabilité éventuelle sur d'autres ordinateurs ou sur d'autres langages de programmation. Les commentaires et les indentations poursuivent les mêmes buts, au détriment parfois de la rapidité d'exécution (du fait de l'interpréteur BASIC). Les personnes soucieuses de performances pourront les éliminer provisoirement. Pour notre part, c'est aussi par souci de performance que nous avons placé les sous-programmes les plus fréquemment appelés en tête et les sous-programmes d'initialisation en fin de programme.

Macao



Pour éviter les crises cardiaques, nous avons limité la durée de ce jeu à çinq minutes au maximum. Il s'agit pour le joueur de faire tourner trois roues pour obtenir une des trois combinaisons gagnantes. S'il n'y réussit pas du premier coup, il peut rejouer une, deux ou les trois roues à la fois. Trois figures identiques valent cent points. Le lot de consolation vaut dix points pour deux figures identiques.

```
1  '                    ***************************
2  '                    *         M A C A O       *
3  '                    ***************************
10 GOSUB 1000                  ' initialisation
20 '
30 WHILE jamais = 0            ' boucle sams fin
40   GOSUB 100
50 WEND
60 '
70 '
90 '--------------------------------------------------------------
91 '                        jeu
92 '                        ---
100 FOR i=1 TO 3: ok(i)=1: NEXT    ' les 3 positions a rejouer
110 '
120 CLS #3: PRINT #3,"Une touche SVP"  :
125 CALL vide : WHILE INKEY$ = "" : WEND
130 GOSUB 2000                     ' 1er tour de roue
135 '
140 CLS #3 : PRINT #3
```

```
141 PRINT #3,"Quelle(s) position(s) rejouez-vous? "
142 '
145 CALL vide :num$ = "" : x$ = "" ' vide les entrees precedentes
147 '
150 WHILE x$<> CHR$(13)               ' simule un INPUT
152   IF x$>="1" AND x$<="3" THEN num$=num$+x$ : PRINT #3,x$;" ";
153   x$=INKEY$
155 WEND
157 '
160 FOR i=1 TO 3: ok(i)= 0: NEXT    ' pas de roues a faire tourner
162 '
165 FOR i=1 TO LEN(num$)
170   FOR j=1 TO 3
180     IF VAL(MID$(num$,i,1))=j THEN ok(j)=1       ' a rejouer ?
190   NEXT j
200 NEXT i
210 '
220 IF ok(1)+ok(2)+ok(3) <> 0 THEN GOSUB 2000      ' 2eme tour
230 '
240 GOSUB 700                                      ' resultat
250 '
260 RETURN
270 '
280 '
490 '-------------------------------------------------------------
491 '     affichage des roues
492 '     ------------------
500 PRINT #1,STRING$(8," ") ;
510 FOR i = 1 TO 3
520   c=c(i)
530   IF c = 224 THEN coul = 3                 ' sourire
540   IF c = 227 OR c = 228 THEN coul = 2     ' coeur ou carreau
550   IF c = 226 OR c = 229 THEN coul = 0     ' pique ou trefle
560   PEN #1,coul
570   PRINT #1,CHR$(c);" ";
580 NEXT
590 '
600 PRINT #1
610 '
620 RETURN
630 '
640 '
690 '-------------------------------------------------------------
691 '        resultat
692 '        --------
700 plus = 0
710 IF c(1)=c(2) OR c(1)=c(3) OR c(2)=c(3) THEN plus = 10
720 IF c(1)=c(2) AND c(1)=c(3)               THEN plus = 100
730 score = score + plus
740 PEN #4,2 : PRINT #4,score
750 RETURN
760 '
770 '
780 '
990  '------------------------------------------------------------
991  '      initialisation
992  '      --------------
1000 INK 0,0 : INK 1,1                   ' 0:noir 1:bleu
1005 INK 2,6 : INK 6,26,6                ' 2:rouge vif 6:blanc/rouge
1010 INK 3,24,11 :INK 4,9 : INK 5,12    ' 3:jaune/bleu 4:vert 5:jaune
1020 BORDER 5
1030 PEN 3
1040 '
1050 MODE 0
1060 '
```

```
1070 WINDOW #6,10,17,2,2                ' score :
1080 WINDOW #1,7,13,7,10                ' roues
1090 WINDOW #2,7,13,12,12               ' numero des roues
1100 WINDOW #3,2,19,20,25               ' commandes
1110 WINDOW #4,17,25,2,3                ' total
1120 WINDOW #5,1,8,2,2                  ' temps qui reste
1125 '
1130 PAPER 0 :PAPER #1,4 :PAPER #3,4
1135 CLS:CLS #1:CLS #2:CLS #3:CLS #4:CLS #5:CLS #6
1140 PEN #6,2 : PRINT #6,"Score:"
1150 PRINT #2," 1 2 3"
1155 FOR i=1 TO 3 : c(i) = 224 : NEXT : GOSUB 500    ' sourire
1160 '
1170 INPUT #3,"Duree du jeu      1 a 5 minutes ";minute
1180 IF minute<1 OR minute>5 THEN GOTO 1170
1190 '
1200 sec = minute * 60
1210 EVERY 50,3  GOSUB 4000              ' bats la seconde
1220 AFTER sec*50 ,0  GOSUB 5000         ' fin
1230 '
1240 vide = &BB03                        ' adresse pour
1250                                     ' reinitialiser le clavier
1260 RETURN
1990 '-------------------------------------------------------------
1991 '     les roues tournent
1992 '     ------------------
2000 '
2020 '
2030 rep = 10                   ' 10 tours
2040 EVERY 18,1 GOSUB 3000  ' hasard
2050 WHILE rep > 0: SOUND 3,500-20*rep: WEND    ' attente en musique
2060 xx=REMAIN(1)
2070 RETURN
2080 '
2090 '
2990 '-------------------------------------------------------------
2991 '      hasard
2992 '      ------
3000 RANDOMIZE TIME
3010 FOR i=1 TO 3
3020   IF ok(i) = 1 THEN c(i) = 226 + INT(RND(3)*4)
3030   c(i) = MIN ( c(i) , 229 )
3040 NEXT i
3050 '
3060 SOUND 4,0,10,15,,,1         ' cliquetis
3070 '
3080 GOSUB 500                   ' affichage
3090 '
3100 rep = rep - 1               ' un tour de roue en moins
3110 '
3120 RETURN
3130 '
3140 '
3990 '-------------------------------------------------------------
3991 '        temps qui reste
3992 '        ---------------
4000 m = sec \ 60
4010 s = sec MOD 60
4020 PEN #5,4  : IF sec <10 THEN PEN #5,6
4030 PRINT #5 : PRINT #5,m;s
4040 sec = sec - 1
4050 IF sec < 0 THEN xx = REMAIN(3)
4060 RETURN
4070 '
4080 '
```

```
4990 '---------------------------------------------------------
4991 '          fin
4992 '          ---
5000 CLS #3 : PRINT #3,"Merci de votre visite"
5010 PRINT #3,"A bientot"
5020 CLEAR
5030 WHILE INKEY$ = "" : WEND
5040 END
```

Commentaires

L'affichage du temps qui reste à jouer est géré par interruption (lignes 121Ø et 4ØØØ à 4Ø6Ø). La fin du jeu arrive à n'importe quelle phase du jeu au bout du temps imparti (lignes 122Ø et 5ØØØ à 5Ø4Ø).

Le comportement aléatoire des roues est assuré par les lignes 3ØØØ et 3Ø2Ø. Les valeurs affichées dépendent uniquement du moment où le joueur appuie sur une touche, ce qui est impossible à prévoir au tiers de seconde près (TIME varie toutes les 3ØØ millisecondes).

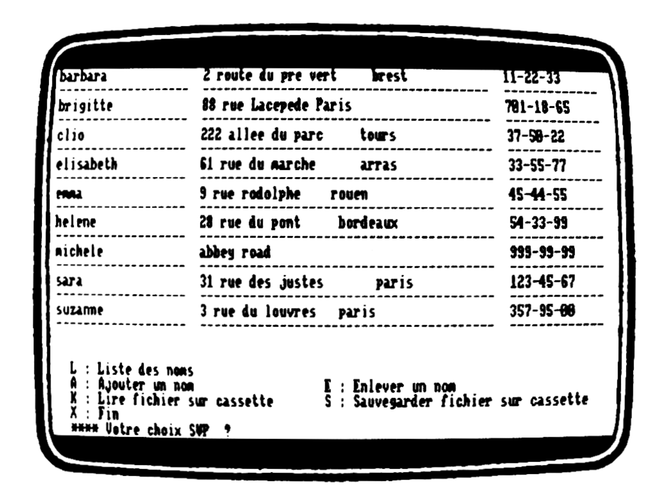
Variations

Il est possible d'augmenter le nombre de roues (r) ou le nombre de figures (f). Il y a alors f^r combinaisons possibles. Ainsi pour trois roues et quatre figures, il y a 64 (4^3) tirages possibles ; pour quatre roues et six figures il y en a 1 296.

Le nombre de tours de roues est fixé à la ligne 2Ø3Ø. Vous pouvez l'augmenter ou le diminuer à volonté. Il n'en est pas de même pour le temps entre deux affichages de roues. En effet, si, à la ligne 2Ø4Ø, vous remplacez la valeur 18 par la valeur 15, les roues ne s'arrêteront pas de tourner. Cela provient du fait que le temps d'exécution du sous-programme d'interruption (3ØØØ-312Ø) est supérieur à 3ØØ millisecondes (15 × 2Ø). Les demandes d'interruption s'"entassent" donc et le BASIC ne pourra jamais exécuter la ligne 2Ø6Ø qui arrête l'appel du sous-programme d'interruption qui fait tourner les roues. Ceci confirme la règle qui veut qu'un sous-programme d'interruption périodique dure moins longtemps que l'intervalle de temps entre deux activations de ce sous-programme.

Annuaire

Est-il plus pratique d'avoir son annuaire personnel sur un carnet ou sur support magnétique ? La réponse dépend de la taille et de l'utilisation de cet annuaire. Pour vous laisser le choix, nous vous proposons un petit gestionnaire de fichier. Les fonctionnalités principales sont présentes : lecture, ajout et retrait d'un nom, lecture et écriture sur bande magnétique.




```
1 '                    *******************
2 '                    * A N N U A I R E *
3 '                    *******************
4 '
10 GOSUB 3000                 ' initilisation
20 '
30 WHILE x$ <> "X"
50     GOSUB 3500              ' choix
60     x$ = UPPER$(x$)
70     IF x$="L" THEN GOSUB 1000    ' lire
80     IF x$="A" THEN GOSUB 1200    ' ajouter un nom
90     IF x$="E" THEN GOSUB 1400    ' effacer un nom
100    IF x$="K" THEN GOSUB 1600    ' lire        fichier sur cassette
110    IF x$="S" THEN GOSUB 1800    ' sauvegarder fichier sur cassette
120 WEND
130 '
140 CLS : END
150 '
```

```
990  '-----------------------------------------------------------------
991  ' lire la liste des noms
992  ' --------------------
1000 CLS : LOCATE #1,1,1 :LOCATE #2,1,1 :LOCATE #3,1,1
1010 '
1020 FOR i = 0 TO n
1025   mx = 1      ' nombre maximum de lignes pour ce nom
1030   PRINT #1,n$(i)     ' nom
1040    l = LEN(n$(i)) :hn=1+INT(l/ln)   :mx=MAX(mx,hn)
1050   PRINT #2,a$(i)     ' addresse
1060    l = LEN(a$(i)) :ha=1+INT(l/la)   :mx=MAX(mx,ha)
1070   PRINT #3,t$(i)     ' telephone
1080    l = LEN(t$(i)) :ht=1+INT(l/lt)   :mx=MAX(mx,ht)
1085 '
1090   FOR j=hn TO mx-1 :PRINT#1 :NEXT
1095   PRINT#1,STRING$(ln-1,"-")
1100   FOR j=ha TO mx-1 :PRINT#2 :NEXT
1105   PRINT#2,STRING$(la-1,"-")
1110   FOR j=ht TO mx-1 :PRINT#3 :NEXT
1115   PRINT#3,STRING$(lt-1,"-")
1117 '
1120 NEXT
1130 '
1140 RETURN
1190 '-----------------------------------------------------------------
1191 '  ajouter un nom
1192 '  -------------
1200 INPUT #4,"nom              ";n$
1210 INPUT #4,"adresse          ";a$
1220 INPUT #4,"telephone        ";t$
1230 n = n + 1                              ' un nom supplementaire
1240 n$(n)=n$ : a$(n)=a$ : t$(n)=t$        ' memorisation
1250 '
1260 GOSUB 2200                             ' tri
1270 '
1280 RETURN
1390 '-----------------------------------------------------------------
1391 '   enlever un nom
1392 '   --------------
1400 INPUT #4,"nom a effacer  ";n$
1405 m = -1
1410 FOR i = 0 TO n
1420   IF n$ = n$(i) THEN m = i : GOTO 1440    ' nom trouve
1430 NEXT i
1440   IF m = -1  THEN PRINT #4,"nom inconnu" : GOTO 1490
1450   n$(m) ="zzzzzz"    ' nom trouve remplace par zzzzzz
1460   GOSUB 2200         ' tri
1470   n = n - 1          ' un nom en moins
1480 '
1490 RETURN
1590 '-----------------------------------------------------------------
1591 '  lire le fichier sur cassette
1592 '  ----------------------------
1600 INPUT #4,"Lecture (Play) puis nom du fichier";f$
1610 OPENIN "!"+f$
1620 INPUT #9,n      ' nombre de personnes + 1
1630 FOR i = 0 TO n
1640   INPUT #9,n$(i) : INPUT #9,a$(i) : INPUT #9,t$(i)
1650 NEXT i
1660 CLOSEIN
1670 RETURN
1680 '
1690 '
1790 '-----------------------------------------------------------------
1791 '  sauvegarder le fichier sur cassette
```

```
1792 '  ------------------------------------
1800 INPUT #4,"Enregistrement (Rec-Play) puis nom du fichier";f$
1810 OPENOUT "!"+f$
1820 PRINT #9,n          ' nombre de personnes + 1
1830 FOR i = 0 TO n
1840  PRINT #9,n$(i) :PRINT #9,a$(i) :PRINT #9,t$(i)
1850 NEXT i
1860 CLOSEOUT
1870 RETURN
1880 '
1890 '
2190 '------------------------------------------------------------------
2191 '   tri simple
2192 '   ----------
2200 FOR i = 0 TO n-1
2230   FOR j = 1 TO n-i
2240     IF n$(j) > n$(j-1)  GOTO 2280
2250     n$=n$(j): n$(j)=n$(j-1): n$(j-1)=n$   ' inversion du nom
2260     a$=a$(j): a$(j)=a$(j-1): a$(j-1)=a$   ' inversion de l'adresse
2270     t$=t$(j): t$(j)=t$(j-1): t$(j-1)=t$   ' inversion du telephone
2280   NEXT j
2290 NEXT i
2300 '
2310 RETURN
2990 ' -----------------------------------------------------------------
2991 ' initialisation
2992 ' --------------
3000 INK 0,13 :INK 1,0                   ' noir sur blanc
3010 PAPER 0 : PEN 1 :BORDER 0
3020 MODE 2
3025 '
3030 DIM n$(500),a$(500)                 ' 501 noms et addresses
3032 DIM t$(500)                         ' 501 telephones
3035 '
3040 ln=20 : la=41 : lt=14               ' largeur
3045 '
3050 WINDOW #1,2,ln+1,1,20               ' nom
3060 WINDOW #2,ln+3,ln+la+2,1,20         ' adresse
3070 WINDOW #3,ln+la+4,ln+la+lt+3,1,20   ' telephone
3080 WINDOW #4,3,80,21,25                ' commandes
3090 '
3100 n = -1      ' aucun nom
3110 '
3120 RETURN
3490 ' -----------------------------------------------------------------
3491 '   choix
3492 '   -----
3500 PRINT #4," L : Liste des noms"
3510 PRINT #4," A : Ajouter un nom";
3515 PRINT #4,"                    E : Enlever un nom"
3520 PRINT #4," K : Lire fichier sur cassette";
3525 PRINT #4,"          S : Sauvegarder fichier sur cassette"
3530 PRINT #4," X : Fin "
3540 PRINT #4," **** Votre choix SVP  ? "
3550 '
3555 CALL &BB03
3560 x$=INKEY$ :IF x$="" GOTO 3560
3570 '
3580 RETURN
```

Variations

Le tri proposé, dit tri bulle, est très simple (lignes 2200 à 2300). Vous pouvez le remplacer par un autre, plus rapide, si vos fichiers dépassent une cinquantaine de personnes.

L'affichage de l'annuaire (lignes 1000 à 1140) ne tient pas compte de la hauteur de l'écran. Il est toutefois possible d'utiliser le "break" pour arrêter le défilement et une touche quelconque pour le reprendre. Une gestion plus poussée du "break" (ON BREAK GOSUB) conduira à une plus grande souplesse. Il peut également être intéressant d'afficher uniquement le fichier page par page, c'est-à-dire écran après écran. Une impression sur papier (hard copy) ne demandera pas d'effort particulier en utilisant le canal #8 qui est le canal imprimante.

Le menu de cet annuaire peut s'enrichir d'une fonction de recherche d'un nom ou d'une partie d'une adresse ou d'un numéro de téléphone. La mise à jour d'une adresse ou d'un téléphone peut se faire partiellement, sans avoir à réécrire toutes les coordonnées de la personne.

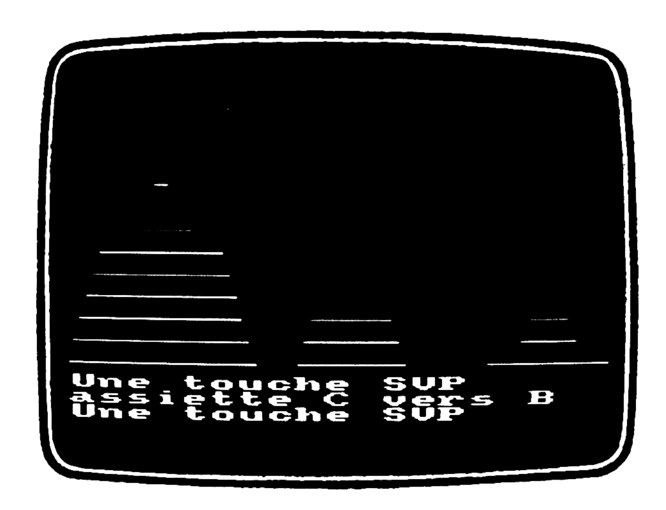
Enfin, si vous voulez que votre fichier reste confidentiel, vous pouvez le coder et le décoder en ajoutant et retranchant une valeur quelconque à chaque caractère ASCII du fichier. Les commandes de codage et de décodage se feront par des touches que vous n'afficherez pas, bien entendu, sur le menu.

Hanoi

Voici un programme qui intéressera sans aucun doute les amateurs de casse-tête. Celui-ci, inventé au siècle dernier par le mathématicien Georges Lucas, est connu sous le nom de Tour de Hanoi.

Cette tour est une pile d'assiettes qu'il faut déplacer d'un plateau

vers un autre. Il y a bien sûr des règles du jeu que nous exposons au long de l'exemple suivant :



Trois plateaux, A, B et C, sont posés sur une table. Sur le plateau A sont empilées six assiettes de tailles différentes, la plus petite en haut, la plus grande en bas. Il s'agit de déplacer ces assiettes vers le plateau C en sachant que :

— on ne peut porter qu'une assiette à la fois,

— une assiette ne peut être posée que sur un plateau vide ou sur une assiette plus grande qu'elle-même.

Nous ne vous donnerons pas la solution car le programme présenté ici le fera bien plus facilement que nous, et pour un nombre d'assiettes à déplacer plus important si vous le désirez.

Utilisation du programme

Après avoir indiqué le nombre d'assiettes, l'origine et la destination de ces assiettes, il ne vous reste plus qu'à appuyer sur une touche du clavier pour que la solution apparaisse. Chaque touche appuyée fait déplacer une assiette jusqu'à la solution finale du casse-tête.

```
1 '             ***************************
2 '             * T O U R   D E   H A N O I *
3 '             ***************************
4 '
10 CLS : CLEAR :  GOSUB 1000          ' initialisation graphique
15 '
20 INPUT #1,"Nombre d'assiettes a deplacer (1 a 15)" ; n
25 IF n < 1 OR n > 15 THEN GOTO 20 ELSE n0 = n
28 '
30 INPUT #1,"De (a,b ou c) ";x$ : x$ = LOWER$(x$)
35 IF x$<>"a" AND x$<>"b" AND x$<>"c" THEN GOTO 30
40 '
50 INPUT #1,"Vers           ";z$ : z$ = LOWER$(z$)
60 IF (z$<>"a" AND z$<>"b" AND z$<>"c") OR z$ = x$  GOTO 50
65 '
70 y=ASC(x$)+ASC(z$)-2*ASC("a") : y = 3-y  ' calcul de y
72 y$ = CHR$(y+ASC("a"))                   ' calcul de y$
75 '
80 GOSUB 1100                    ' dessin initial
85 '
90 GOSUB 200                     ' solution
95 '
100 b$ = INKEY$ : IF b$ = "" THEN GOTO 100 ELSE GOTO 10
110 '
120 '
130 '
190 ' solution pour n assiettes de x$ vers z$ par y$
191 ' ---------------------------------------------
200 IF n = 0 GOTO 470
210  n = n - 1
220     t$ = z$ : z$ = y$ : y$ = t$         ' y$ <--> z$
230     GOSUB 200                           ' appel recursif
240     t$ = y$ : y$ = z$ : z$ = t$         ' y$ <--> z$
290 '
300     PEN #1,15 : PRINT #1,"assiette ";   ' origine
301     PEN #1,8  : PRINT #1,UPPER$(x$);
302     PEN #1,15 : PRINT #1," vers ";      ' destination
303     PEN #1,8  : PRINT #1,UPPER$(z$);
304     PEN #1,15
305     GOSUB 500                           ' visualisation
306                                         ' graphique
310 '
420     t$ = x$ : x$ = y$ : y$ = t$         ' x$ <--> y$
430     GOSUB 200                           ' appel recursif
440     t$ = y$ : y$ = x$ : x$ = t$         ' x$ <--> y$
450  n = n + 1
460 '
470 RETURN
480 '
481 '
482 '
490 ' deplacement graphique de x$ vers z$
491 ' ---------------------------------
500 PRINT #1,"Une touche SVP"
505 attente$ = INKEY$ : IF attente$ = "" THEN GOTO 505
506 '
510 de   = ASC(x$) - ASC("a")     ' a:0 b:1 c:2
512 vers = ASC(z$) - ASC("a")     ' a:0 b:1 c:2
520 '
530 ptr(de) = ptr(de) - 1         ' depilage
```

```
540 ass = pile(ptr(de),de)
545 ssa = n0 - ass + 1
550 '
560 MOVE 107+213*de-ass*larg,h0+(ptr(de)+1)*dis   ' efface ancienne
570 DRAWR 2*ass*larg,0,0                         ' position
580 '
590 pile(ptr(vers),vers) = ass    ' empilage
600 ptr(vers) = ptr(vers) + 1
610 '
620 MOVE 107+213*vers-ass*larg,h0+ptr(vers)*dis  ' nouvelle position
630 DRAWR 2*ass*larg,0,ass                       ' visible
640 '
650 RETURN
660 '
670 '
680 '
990 ' initialisation graphique  x$ : tas initial
991 ' ------------------------  n  : nombre d'assiettes
1000 MODE 0
1005 INK 0,0 : PAPER 0 : BORDER 0 ' papier et bord noir
1010 WINDOW #1,1,20,23,25
1015 '
1020 FOR i=1 TO 15                ' chaque encre a sa couleur
1030   INK i,i+7
1035   PEN #1,i :PRINT #1,"       H A N O I"
1040 NEXT
1050 '
1060 DIM pile(14,2)               ' 3 piles de 15 assiettes
1065 DIM ptr(2)                   ' 3 pointeurs
1070 '
1080 RETURN
1090 '
1091 '
1092 '
1095 ' dessin initial
1096 ' --------------
1100                               ' distance entre assiettes
1110 IF n0<2 THEN dis = 339 ELSE dis = 339/(n0-1)
1120 larg = 107/n0                 ' largeur assiette
1130 h0 = 60 - dis                 ' hauteur minimum
1140 tas = ASC(x$)-ASC("a")        ' a:0  b:1  c:2
1150 '
1160 FOR i = 1 TO n0
1170   j = n0 - i + 1
1180   pile(ptr(tas),tas) = j       ' initialisation pile
1190   ptr(tas) = ptr(tas) + 1
1200   MOVE 107+213*tas-j*larg,h0+i*dis
1210   DRAWR 2*j*larg,0,j           ' dessin assiette
1220 NEXT i
1230 '
1240 RETURN
```

Commentaires

Le mode Ø (ligne 1ØØØ) est celui qui autorise le maximum de couleurs présentes en même temps sur l'écran. L'encrier Ø contient la couleur noire (ligne 1ØØ5) et chacun des quinze autres encriers est rempli d'une encre de couleur différente (lignes 1Ø2Ø à 1Ø4Ø).

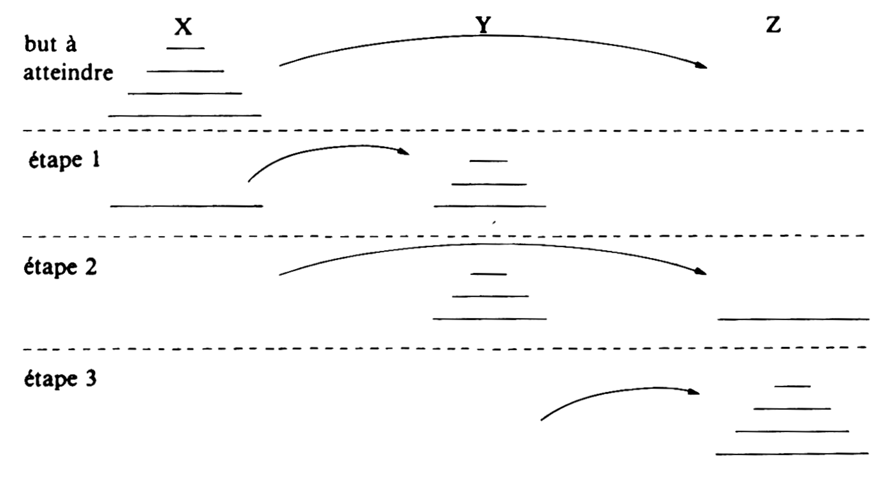
Le fond de l'écran étant noir, l'effacement d'une assiette est réalisé

en la coloriant en noir (ligne 57Ø). La variable ass (ligne 63Ø) détermine la couleur de l'assiette.
L'état des trois plateaux est représenté par des piles au sens classique en informatique de mémoire où le dernier élément introduit est le premier élément ressorti.

La solution du casse-tête réside dans l'appel du sous-programme 2ØØ-47Ø. Cette solution découle de la remarque suivante :

Déplacer n assiettes du plateau **X** vers le plateau **Z** (lignes 2ØØ à 47Ø) équivaut à :

1. **déplacer (n – 1)** assiettes du plateau **X** vers le plateau **Y** (lignes 21Ø à 24Ø),
2. **porter** 1 assiette du plateau **X** vers le plateau **Z** (lignes 3ØØ à 3Ø5),
3. **déplacer (n – 1)** assiettes du plateau **Y** vers le plateau **Z** (lignes 42Ø à 45Ø).



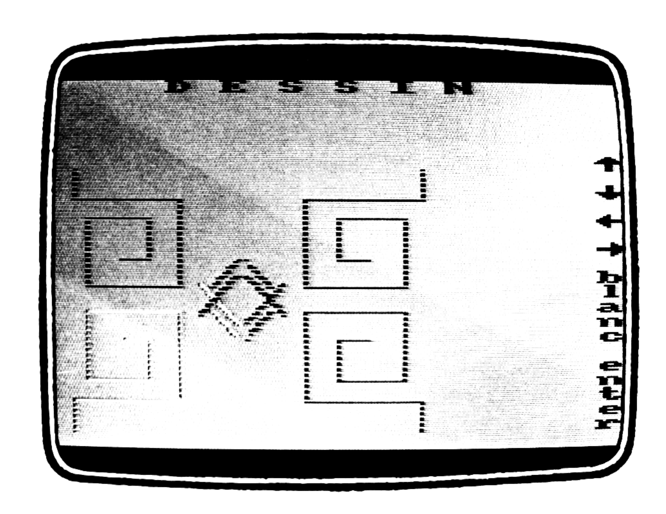
Solution du problème en trois étapes.

La deuxième étape est autorisée par la règle du jeu alors que les deux autres ne le sont pas. L'étape 1 est résolue en appelant le sous-programme 2ØØ-47Ø avec une assiette en moins et en inversant Y et Z (X vers Y et non X vers Z). L'étape 3 a une solution comparable.

Le sous-programme 2ØØ, s'appelant lui-même (lignes 23Ø et 43Ø), est un sous-programme **récursif**. Il faut donc veiller à ce que les variables retrouvent leur état initial après les appels récursifs (lignes 24Ø et 44Ø). La variable **n** n'est pas incrémentée à la ligne 25Ø car elle serait immédiatement décrémentée à la ligne 41Ø.

Narcisse

Il avait une passion folle pour son image. Il se précipita dans les eaux de la fontaine qui lui servait de miroir et donna son nom à une fleur.



Dans un premier temps, vous définissez les axes et les points de symétrie. Les quatre flèches déplacent un point clignotant. La touche [.] fige un point de symétrie. La touche [copy] valide les axes de symétrie. Il peut y avoir jusqu'à quinze points et quinze axes de symétrie. La touche [enter] vous conduit au mode dessin. Les quatre flèches déplacent le curseur. La barre d'espace change alternativement le curseur en gomme ou en crayon.

```
1 '                 ********************
2 '                 * N A R C I S S E *
3 '                 ********************
10 '
20 GOSUB 500             ' initialisation
30 '
35  PRINT #1,CHR$(240);" ";CHR$(241);" ";CHR$(242)      ' mode
36  PRINT #1,;CHR$(243);" ";"copy";" ";".";" ";"enter" ' d'emploi
37 '
40 a$ = INKEY$ : IF a$ = "" GOTO 40 ELSE aa = ASC(a$)
45 '
50 WHILE jamais = 0    ' boucle infinie
60   ON mde+1 GOSUB 1000,2000
```

```
80   a$ = INKEY$ : IF a$ = "" GOTO 80 ELSE aa = ASC(a$)
90 WEND
100 '
110 '
120 '
490 ' initialisation
491 ' --------------
500 MODE 0
510 larg = 600 : haut = 365
520 df1 = 10 : df0 = 3                ' precision des traces
530 DIM a(15),b(15),c(15)             ' 15 droites
540 DIM p0(15),p1(15)                 ' 15 points de symetrie
545 '
550 FOR i = 1 TO 15
560   INK i,i+2                       ' chaque symetrie
570 NEXT i                            ' a sa couleur
575 '
580 INK 0,13 : INK 11,26              ' 0:gris  1:blanc
590 BORDER 0 : PAPER 0 : PEN 1        ' bords noir  fond blanc
600 '
610 WINDOW #1,20,20,3,25
614 WINDOW #2,1,20,1,1
615 PRINT #2,"  N A R C I S S E";
620 '
630 RETURN
640 '
989 '-----------------------------------------------------------------
990 '        mode definition des symetries
991 '        -----------------------------
1000 '
1010 '
1020 '
1030 IF aa=240 THEN y=(y+df1) MOD haut : GOTO 1100  ' H A U T
1040 IF aa=241 THEN y=(y-df1) MOD haut : GOTO 1100  ' B A S
1050 IF aa=242 THEN x=(x-df1) MOD larg : GOTO 1100  ' G A U C H E
1060 IF aa=243 THEN x=(x+df1) MOD larg : GOTO 1100  ' D R O I T E
1070 '
1080 GOTO 1200      ' suite des tests
1090 '
1100 IF x<0 THEN x= x+larg
1110 IF y<0 THEN y= y+haut
1160 PLOT ax,ay,0 : ax=x : ay=y      ' efface ancien point
1165 PLOT x,y,4                      ' marque nouveau point
1170 GOTO 1700     ' return
1180 '
1200 IF aa <> 224 GOTO 1400                        ' C O P Y ( axe )
1205   IF nsd <= 14 THEN ax=0: ay=0 ' ne pas effacer ancien point
1210   IF premier=0
       THEN x0=x:y0=y: premier=1: GOTO 1700 ' 1er point de droite
1220      x1=x : y1=y                       ' 2eme point de droite
1230      nsd = nsd + 1
1235      IF nsd > 15 GOTO 1700             ' numero de sym/droite
1240      MOVE x0,y0                ' dessin droite
1250      DRAW x1,y1,nsd
1260 '
1270      GOSUB 3000'               ' calcul a,b,c dans ax+by+c = 0
1280 '
1290      a(nsd)=a : b(nsd)=b : c(nsd)=c
1300 '
1310    premier = 0
1320 GOTO 1700     ' return
1330 '
1400 IF a$ <> "." GOTO 1500         ' P O I N T (symetrie/point)
1410   nsp = nsp + 1
1415   IF nsp > 15 GOTO 1700        ' numero de symetrie/point
```

```
1420 '
1425   ax = 0 : ay = 0                    ' ne pas effacer ancien point
1430   p0(nsp) = x                        ' abscisse point de symetrie
1440   p1(nsp) = y                        ' ordonnee point de symetrie
1450   PLOT x,y,nsp
1460 '
1470 '
1500 IF aa <> 13 GOTO 1700               ' E N T E R ( --> mode dessin )
1510   mde = 1 - mde                      ' changement de mode
1520   premier = 0                        ' premier point d'une droite
1522   x=0 : y=0                          ' coordonnees
1525   CLS
1530    PRINT#1,CHR$(240);" ";CHR$(241);" ";CHR$(242);   ' mode
1532    PRINT#1," ";CHR$(243);" ";"blanc";" ";"enter"    ' d'emploi
1535 PRINT #2,"     D E S S I N ";
1540 GOTO 1700
1550 '
1560 '
1700 RETURN
1710 '
1720 '
1989 '-------------------------------------------------------------------
1990 '        mode dessin
1991 '        -----------
2000 '
2030 IF aa = 240 THEN y = (y+df0) : GOTO 2100    ' H A U T
2040 IF aa = 241 THEN y = (y-df0) : GOTO 2100    ' B A S
2050 IF aa = 242 THEN x = (x-df0) : GOTO 2100    ' G A U C H E
2060 IF aa = 243 THEN x = (x+df0) : GOTO 2100    ' D R O I T E
2080 GOTO 2400     ' suite des tests
2090 '
2100 '
2120 '
2130   IF inv=0 THEN PLOT ax,ay,0         ' efface ancien point
2135                                      ' si mode invisible
2140   ax = x : ay = y                    ' ancien point
2160   PLOT x,y,4                         ' nouveau point dessine
2170 '
2180   FOR i = 1 TO nsd                   ' symetries/droites
2190     a=a(i) : b=b(i) : c =c(i)        ' ax+by+c = 0
2200     x1 = (-2*a*(c+b*y)+(b^2-a^2)*x) / (a^2+b^2) ' abscisse image
2210     y1 = (-2*b*(c+a*x)+(a^2-b^2)*y) / (a^2+b^2) ' ordonnee image
2220     PLOT x1,y1,i*inv                 ' point invisible au depart
2230   NEXT i
2240 '
2250   FOR i = 1 TO nsp                   ' symetries/points
2260     x1 = 2*p0(i)  - x                ' abscisse image
2270     y1 = 2*p1(i)  - y                ' ordonnee image
2275     x1 = x1 MOD larg  :  y1 = y1 MOD haut
2280     PLOT x1,y1,i*inv                 ' dessin du point symetrique
2290   NEXT i
2300 '
2350 '
2400 IF a$ = " "
     THEN inv=1-inv : GOTO 2700   ' E S P A C E (visible/invisible)
2420 '
2430 '
2500 IF aa <> 13 GOTO 2700               ' E N T E R ( --> mode definition )
2510   mde = 1 - mde                      ' changement de mode
2520   CLG
2540   PRINT#1,CHR$(240);" ";CHR$(241);" ";CHR$(242);
2542   PRINT#1,CHR$(243);" ";"copy";" ";".";" ";"enter"
2545 PRINT #2,"S Y M E T R I E S";
2550 '
```

```
2700 RETURN
2885 '
2886 '
2887 '
2990 ' calcul des coefficients a,b,c (ax+by+c=0) a partir de deux points
2991 ' -----------------------------
3000 IF y0 = y1 THEN a=0 :b=1 :c=-y0  :GOTO 3060 ' axe horizontal
3010 '
3020     a = 1
3030     b = (x1-x0)/(y0-y1)
3040     c = -x0- b*y0
3050 '
3060 RETURN
4000 a$=INKEY$:IF a$=""GOTO 4000 ELSE PRINT a$,ASC(a$):GOTO 4000
```

Commentaires

Les tableaux a(15), b(15) et c(15) définissent les quinze droites de symétrie, d'équation ax + by + c = Ø. Les tableaux pØ(15) et p1(15) contiennent les coordonnées des points de symétrie. Le calcul des points image est fait aux lignes 22ØØ-221Ø et 226Ø-227Ø. La variable inv vaut 1 pour la position crayon et Ø pour la position gomme.

Variations

Si vous possédez un Joystick, vous penserez sûrement à modifier les lignes 1Ø3Ø à 1Ø6Ø et 2Ø3Ø à 2Ø6Ø.

Les artistes soucieux de conserver leur création pourront garder la mémoire écran sur cassette ou sur disquette.

La définition du dessin peut être améliorée au maximum, au détriment du temps de calcul des symétries, en affectant la valeur 1 à la constante dfØ (ligne 12Ø). Si la couleur vous indiffère, le mode 2 vous donnera la plus grande précision possible pour le tracé des dessins.

Annexes

Annexe 1
Liste des instructions du Z8Ø classées par ordre alphabétique

Les octets marqués par . . sont à compléter par l'opérande de l'instruction, c'est-à-dire la valeur N ou NN, l'adresse (NN) ou le déplacement d.

Exemple :
AND (IX + &Ø5). Se reporter à l'instruction
AND (IX + d) de code **DD A6 . .**
Remplacer . . par **Ø5**
L'instruction est **DD A6 Ø5**

Code	Instruction	Code	Instruction	Code	Instruction
8E	ADC A,(HL)	FD09	ADD IY,BC	CB4D	BIT 1,L
DD8E••	ADC A,(IX+d)	FD19	ADD IY,DE	CB56	BIT 2,(HL)
FD8E••	ADC A,(IY+d)	FD29	ADD IY,IY	DDCB••56	BIT 2,(IX+d)
8F	ADC A,A	FD39	ADD IY,SP	FDCB••56	BIT 2,(IY+d)
88	ADC A,B	A6	AND (HL)	CB57	BIT 2,A
89	ADC A,C	DDA6••	AND (IX+d)	CB50	BIT 2,B
8A	ADC A,D	FDA6••	AND (IY+d)	CB51	BIT 2,C
8B	ADC A,E	A7	AND A	CB52	BIT 2,D
8C	ADC A,H	A0	AND B	CB53	BIT 2,E
8D	ADC A,L	A1	AND C	CB54	BIT 2,H
CE••	ADC A,N	A2	AND D	CB55	BIT 2,L
ED4A	ADC HL,BC	A3	AND E	CB5E	BIT 3,(HL)
ED5A	ADC HL,DE	A4	AND H	DDCB••5E	BIT 3,(IX+d)
ED6A	ADC HL,HL	A5	AND L	FDCB••5E	BIT 3,(IY+d)
ED7A	ADC HL,SP	E6••	AND N	CB5F	BIT 3,A
86	ADD A,(HL)	CB46	BIT 0,(HL)	CB58	BIT 3,B
DD86••	ADD A,(IX+d)	DDCB••46	BIT 0,(IX+d)	CB59	BIT 3,C
FD86••	ADD A,(IY+d)	FDCB••46	BIT 0,(IY+d)	CB5A	BIT 3,D
87	ADD A,A	CB47	BIT 0,A	CB5B	BIT 3,E
80	ADD A,B	CB40	BIT 0,B	CB5C	BIT 3,H
81	ADD A,C	CB41	BIT 0,C	CB5D	BIT 3,L
82	ADD A,D	CB42	BIT 0,D	CB66	BIT 4,(HL)
83	ADD A,E	CB43	BIT 0,E	DDCB••66	BIT 4,(IX+d)
84	ADD A,H	CB44	BIT 0,H	FDCB••66	BIT 4,(IY+d)
85	ADD A,L	CB45	BIT 0,L	CB67	BIT 4,A
C6••	ADD A,N	CB4E	BIT 1,(HL)	CB60	BIT 4,B
09	ADD HL,BC	DDCB••4E	BIT 1,(IX+d)	CB61	BIT 4,C
19	ADD HL,DE	FDCB••4E	BIT 1,(IY+d)	CB62	BIT 4,D
29	ADD HL,HL	CB4F	BIT 1,A	CB63	BIT 4,E
39	ADD HL,SP	BC48	BIT 1,B	CB64	BIT 4,H
DD09	ADD IX,BC	CB49	BIT 1,C	CB65	BIT 4,L
DD19	ADD IX,DE	CB4A	BIT 1,D	CB6E	BIT 5,(HL)
DD29	ADD IX,IX	CB4B	BIT 1,E	DDCB••6E	BIT 5,(IX+d)
DD39	ADD IX,SP	CB4C	BIT 1,H	FDCB••6E	BIT 5,(IY+d)
CB6F	BIT 5,A	DD2B	DEC IX	71	LD (HL),C
CB68	BIT 5,B	FD2B	DEC IY	72	LD (HL),D
CB69	BIT 5,C	2D	DEC L	73	LD (HL),E
CB6A	BIT 5,D	3B	DEC SP	74	LD (HL),H
CB6B	BIT 5,E	F3	DI	75	LD (HL),L
CB6C	BIT 5,H	10••	DJNZ DIS	36••	LD (HL),N
CB6D	BIT 5,L	FB	EI	DD77••	LD (IX+d),A
CB76	BIT 6,(HL)	E3	EX (SP),HL	DD70••	LD (IX+d),B
DDCB••76	BIT 6,(IX+d)	DDE3	EX (SP),IX	DD71••	LD (IX+d),C
FDCB••76	BIT 6,(IY+d)	FDE3	EX (SP),IY	DD72••	LD (IX+d),D
CB77	BIT 6,A	08	EX AF,AF'	DD73••	LD (IX+d),E
CB70	BIT 6,B	EB	EX DE,HL	DD74••	LD (IX+d),H
CB71	BIT 6,C	D9	EXX	DD75••	LD (IX+d),L
CB72	BIT 6,D	76	HALT	DD36••••	LD (IX+d),N
CB73	BIT 6,E	ED46	IM 0	FD77••	LD (IY+d),A
CB74	BIT 6,H	ED56	IM 1	FD70••	LD (IY+d),B
CB75	BIT 6,L	ED5E	IM 2	FD71••	LD (IY+d),C
CB7E	BIT 7,(HL)	ED78	IN A,(C)	FD72••	LD (IY+d),D
DDCB••7E	BIT 7,(IX+d)	DB••	IN A,(N)	FD73••	LD (IY+d),E
FDCB••7E	BIT 7,(IY+d)	ED40	IN B,(C)	FD74••	LD (IY+d),H
CB7F	BIT 7,A	ED48	IN C,(C)	FD75••	LD (IY+d),L
CB78	BIT 7,B	ED50	IN D,(C)	FD36••••	LD (IY+d),N
CB79	BIT 7,C	ED58	IN E,(C)	32••••	LD (NN),A
CB7A	BIT 7,D	ED60	IN H,(C)	ED43••••	LD (NN),BC
CB7B	BIT 7,E	ED68	IN L,(C)	ED53••••	LD (NN),DE
CB7C	BIT 7,H	34	INC (HL)	22••••	LD (NN),HL
CB7D	BIT 7,L	DD34••	INC (IX+d)	DD22••••	LD (NN),IX

Code	Instruction	Code	Instruction	Code	Instruction
DC••••	CALL C,NN	FD34••	INC (IY+d)	FD22••••	LD (NN),IY
FC••••	CALL M,NN	3C	INC A	ED73••••	LD (NN),SP
D4••••	CALL NC,NN	04	INC B	0A	LD A,(BC)
CD••••	CALL NN	03	INC BC	1A	LD A,(DE)
C4••••	CALL NZ,NN	0C	INC C	7E	LD A,(HL)
F4••••	CALL P,NN	14	INC D	DD7E••	LD A,(IX+d)
EC••••	CALL PE,NN	13	INC DE	FD7E••	LD A,(IY+d)
E4••••	CALL PO,NN	1C	INC E	3A••••	LD A,(NN)
CC••••	CALL Z,NN	24	INC H	7F	LD A,A
3F	CCF	23	INC HL	78	LD A,B
BE	CP (HL)	DD23	INC IX	79	LD A,C
DDBE••	CP (IX+d)	FD23	INC IY	7A	LD A,D
FDBE••	CP (IY+d)	2C	INC L	7B	LD A,E
BF	CP A	33	INC SP	7C	LD A,H
B8	CP B	EDAA	IND	ED57	LD A,I
B9	CP C	EDBA	INDR	7D	LD A,L
BA	CP D	EDA2	INI	3E••	LD A,N
BB	CP E	EDB2	INIR	46	LD B,(HL)
BC	CP H	E9	JP (HL)	DD46••	LD B,(IX+d)
BD	CP L	DDE9	JP (IX)	FD46••	LD B,(IY+d)
FE••	CP N	FDE9	JP (IY)	47	LD B,A
EDA9	CPD	DA••••	JP C,NN	40	LD B,B
EDB9	CPDR	FA••••	JP M,NN	41	LD B,C
EDA1	CPI	D2••••	JP NC,NN	42	LD B,D
EDB1	CPIR	C3••••	JP NN	43	LD B,E
2F	CPL	C2••••	JP NZ,NN	44	LD B,H
27	DAA	F2••••	JP P,NN	45	LD B,L
35	DEC (HL)	EA••••	JP PE,NN	06••	LD B,N
DD35••	DEC (IX+d)	E2••••	JP PO,NN	ED4B••••	LD BC,(NN)
FD35••	DEC (IY+d)	CA••••	JP Z,NN	01••••	LD BC,NN
3D	DEC A	38••	JR C,DIS	4E	LD C,(HL)
05	DEC B	18••	JR DIS	DD4E••	LD C,(IX+d)
0B	DEC BC	30••	JR NC,DIS	FD4E••	LD C,(IY+d)
0D	DEC C	20••	JR NZ,DIS	4F	LD C,A
15	DEC D	28••	JR Z,DIS	48	LD C,B
1B	DEC DE	02	LD (BC),A	49	LD C,C
1D	DEC E	12	LD (DE),A	4A	LD C,D
25	DEC H	77	LD (HL),A	4B	LD C,E
2B	DEC HL	70	LD (HL),B	4C	LD C,H
4D	LD C,L	DDB6••	OR (IX+d)	CB9F	RES 3,A
0E••	LD C,N	FDB6••	OR (IY+d)	CB98	RES 3,B
56	LD D,(HL)	B7	OR A	CB99	RES 3,C
DD56••	LD D,(IX+d)	B0	OR B	CB9A	RES 3,D
FD56••	LD D,(IY+d)	B1	OR C	CB9B	RES 3,E
57	LD D,A	B2	OR D	CB9C	RES 3,H
50	LD D,B	B3	OR E	CB9D	RES 3,L
51	LD D,C	B4	OR H	CBA6	RES 4,(HL)
52	LD D,D	B5	OR L	DDCB••A6	RES 4,(IX+d)
53	LD D,E	F6••	OR N	FDCB••A6	RES 4,(IY+d)
54	LD D,H	EDBB	OTDR	CBA7	RES 4,A
55	LD D,L	EDB3	OTIR	CBA0	RES 4,B
1620	LD D,N	ED79	OUT (C),A	CBA1	RES 4,C
ED5B••••	LD DE,(NN)	ED41	OUT (C),B	CBA2	RES 4,D
11••••	LD DE,NN	ED49	OUT (C),C	CBA3	RES 4,E
5E	LD E,(HL)	ED51	OUT (C),D	CBA4	RES 4,H
DD5E••	LD E,(IX+d)	ED59	OUT (C),E	CBA5	RES 4,L
FD5E••	LD E,(IY+d)	ED61	OUT (C),H	CBAE	RES 5,(HL)
5F	LD E,A	ED69	OUT (C),L	DDCB••AE	RES 5,(IX+d)
58	LD E,B	D3••	OUT (N),A	FDCB••AE	RES 5,(IY+d)
59	LD E,C	EDAB	OUTD	CBAF	RES 5,A
5A	LD E,D	EDA3	OUTI	CBA8	RES 5,B

Code	Instruction	Code	Instruction	Code	Instruction
5B	LD E,E	F1	POP AF	CBA9	RES 5,C
5C	LD E,H	C1	POP BC	CBAA	RES 5,D
5D	LD E,L	D1	POP DE	CBAB	RES 5,E
1E••	LD E,N	E1	POP HL	CBAC	RES 5,H
66	LD H,(HL)	DDE1	POP IX	CBAD	RES 5,L
DD66••	LD H,(IX+d)	FDE1	POP IY	CBB6	RES 6,(HL)
FD66••	LD H,(IY+d)	F5	PUSH AF	DDCB••B6	RES 6,(IX+d)
67	LD H,A	C5	PUSH BC	FDCB••B6	RES 6,(IY+d)
60	LD H,B	D5	PUSH DE	CBB7	RES 6,A
61	LD H,C	E5	PUSH HL	CBB0	RES 6,B
62	LD H,D	DDE5	PUSH IX	CBB1	RES 6,C
63	LD H,E	FDE5	PUSH IY	CBB2	RES 6,D
64	LD H,H	CB86	RES 0,(HL)	CBB3	RES 6,E
65	LD H,L	DDCB••86	RES 0,(IX+d)	CBB4	RES 6,H
26••	LD H,N	FDCB••86	RES 0,(IY+d)	CBB5	RES 6,L
2A••••	LD HL,(NN)	CB87	RES 0,A	CBBE	RES 7,(HL)
21••••	LD HL,NN	CB80	RES 0,B	DDCB••BE	RES 7,(IX+d)
ED47	LD I,A	CB81	RES 0,C	FDCB••BE	RES 7,(IY+d)
DD2A••••	LD IX,(NN)	CB82	RES 0,D	CBBF	RES 7,A
DD21••••	LD IX,NN	CB83	RES 0,E	CBB8	RES 7,B
FD2A••••	LD IY,(NN)	CB84	RES 0,H	CBB9	RES 7,C
FD21••••	LD IY,NN	CB85	RES 0,L	CBBA	RES 7,D
6E	LD L,(HL)	CB8E	RES 1,(HL)	CBBB	RES 7,E
DD6E••	LD L,(IX+d)	DDCB••8E	RES 1,(IX+d)	CBBC	RES 7,H
FD6E••	LD L,(IY+d)	FDCB••8E	RES 1,(IY+d)	CBBD	RES 7,L
6F	LD L,A	CB8F	RES 1,A	C9	RET
68	LD L,B	CB88	RES 1,B	D8	RET C
69	LD L,C	CB89	RES 1,C	F8	RET M
6A	LD L,D	CB8A	RES 1,D	D0	RET NC
6B	LD L,E	CB8B	RES 1,E	C0	RET NZ
6C	LD L,H	CB8C	RES 1,H	F0	RET P
6D	LD L,L	CB8D	RES 1,L	E8	RET PE
2E••	LD L,N	CB96	RES 2,(HL)	E0	RET PO
ED••••	LD SP,(NN)	DDCB••96	RES 2,(IX+d)	C8	RET Z
F9	LD SP,HL	FDCB••96	RES 2,(IY+d)	ED4D	RETI
DDF9	LD SP,IX	CB97	RES 2,A	ED45	RETN
FDF9	LD SP,IY	CB90	RES 2,B	CB16	RL (HL)
31••••	LD SP,NN	CB91	RES 2,C	DDCB••16	RL (IX+d)
EDA8	LDD	CB92	RES 2,D	FDCB••16	RL (IY+d)
EDB8	LDDR	CB93	RES 2,E	CB17	RL A
EDA0	LDI	CB94	RES 2,H	CB10	RL B
EDB0	LDIR	CB95	RES 2,L	CB11	RL C
ED44	NEG	CB9E	RES 3,(HL)	CB12	RL D
00	NOP	DDCB••9E	RES 3,(IX+d)	CB13	RL E
B6	OR (HL)	FDCB••9E	RES 3,(IY+d)	CBFE	SET 7,(HL)
CB14	RL H	CBC0	SET 0,B	DDCB••FE	SET 7,(IX+d)
CB15	RL L	CBC1	SET 0,C	FDCB••FE	SET 7,(IY+d)
17	RLA	CBC2	SET 0,D	CBFF	SET 7,A
CB06	RLC (HL)	CBC3	SET 0,E	CBF8	SET 7,B
DDCB••06	RLC (IX+d)	CBC4	SET 0,H	CBF9	SET 7,C
FDCB••06	RLC (IY+d)	CBC5	SET 0,L	CBFA	SET 7,D
CB07	RLC A	CBCE	SET 1,(HL)	CBFB	SET 7,E
CB00	RLC B	DDCB••CE	SET 1,(IX+d)	CBFC	SET 7,H
CB01	RLC C	FDCB••CE	SET 1,(IY+d)	CBFD	SET 7,L
CB02	RLC D	CBCF	SET 1,A	CB26	SLA (HL)
CB03	RLC E	CBC8	SET 1,B	DDCB••26	SLA (IX+d)
CB04	RLC H	CBC9	SET 1,C	FDCB••26	SLA (IY+d)
CB05	RLC L	CBCA	SET 1,D	CB27	SLA A
07	RLCA	CBCB	SET 1,E	CB20	SLA B
ED6F	RLD	CBCC	SET 1,H	CB21	SLA C
CB1E	RR (HL)	CBCD	SET 1,L	CB22	SLA D
DDCB••1E	RR (IX+d)	CBD6	SET 2,(HL)		

Code	Instruction
FDCB..1E	RR (IY + d)
CB1F	RR A
CB18	RR B
CB19	RR C
CB1A	RR D
CB1B	RR E
CB1C	RR H
CB1D	RR L
1F	RRA
CB0E	RRC (HL)
DDCB..0E	RRC (IX + d)
FDCB..0E	RRC (IY + d)
CB0F	RRC A
CB08	RRC B
CB09	RRC C
CB0A	RRC D
CB0B	RRC E
CB0C	RRC H
CB0D	RRC L
0F	RRCA
ED67	RRD
C7	RST 0
D7	RST 10H
DF	RST 18H
E7	RST 20H
EF	RST 28H
F7	RST 30H
FF	RST 38H
CF	RST 8
9E	SBC A, (HL)
DD9E..	SBC A, (IX + d)
FD9E..	SBC A, (IY + d)
9F	SBC A, A
98	SBC A, B
99	SBC A, C
9A	SBC A, D
9B	SBC A, E
9C	SBC A, H
9D	SBC A, L
DE..	SBC A, N
ED42	SBC HL, BC
ED52	SBC HL, DE
ED62	SBC HL, HL
ED72	SBC HL, SP
37	SCF
CBC6	SET 0, (HL)
DDCB..C5	SET 0, (IX + d)
FDCB..C6	SET 0, (IY + d)
CBC7	SET 0, A
DDCB..D6	SET 2, (IX + d)
FDCB..D6	SET 2, (IY + d)
CBD7	SET 2, A
CBD0	SET 2, B
CBD1	SET 2, C
CBD2	SET 2, D
CBD3	SET 2, E
CBD4	SET 2, H
CBD5	SET 2, L
CBD8	SET 3, B
CBDE	SET 3, (HL)
DDCB..DE	SET 3, (IX + d)
FDCB..DE	SET 3, (IY + d)
CBDF	SET 3, A
CBD9	SET 3, C
CBDA	SET 3, D
CBDB	SET 3, E
CBDC	SET 3, H
CBDD	SET 3, L
CBE6	SET 4, (HL)
DDCB..E6	SET 4, (IX + d)
FDCB..E6	SET 4, (IY + d)
CBE7	SET 4, A
CBE0	SET 4, B
CBE1	SET 4, C
CBE2	SET 4, D
CBE3	SET 4, E
CBE4	SET 4, H
CBE5	SET 4, L
CBEE	SET 5, (HL)
DDCB..EE	SET 5, (IX + d)
FDCB..EE	SET 5, (IY + d)
CBEF	SET 5, A
CBE8	SET 5, B
CBE9	SET 5, C
CBEA	SET 5, D
CBEB	SET 5, E
CBEC	SET 5, H
CBED	SET 5, L
CBF6	SET 6, (HL)
DDCB..F6	SET 6, (IX + d)
FDCB..F6	SET 6, (IY + d)
CBF7	SET 6, A
CBF0	SET 6, B
CBF1	SET 6, C
CBF2	SET 6, D
CBF3	SET 6, E
CBF4	SET 6, H
CBF5	SET 6, L
CB23	SLA E
CB24	SLA H
CB25	SLA L
CB2E	SRA (HL)
DDCB..2E	SRA (IX + d)
FDCB..2E	SRA (IY + d)
CB2F	SRA A
CB28	SRA B
CB29	SRA C
CB2A	SRA D
CB2B	SRA E
CB2C	SRA H
CB2D	SRA L
CB3E	SRL (HL)
DDCB..3E	SRL (IX + d)
FDCB..3E	SRL (IY + d)
CB3F	SRL A
CB38	SRL B
CB39	SRL C
CB3A	SRL D
CB3B	SRL E
CB3C	SRL H
CB3D	SRL L
96	SUB (HL)
DD96..	SUB (IX + d)
FD96..	SUB (IY + d)
97	SUB A
90	SUB B
91	SUB C
92	SUB D
93	SUB E
94	SUB H
95	SUB L
D6..	SUB N
AE	XOR (HL)
DDAE..	XOR (IX + d)
FDAE..	XOR (IY + d)
AF	XOR A
A8	XOR B
A9	XOR C
AA	XOR D
AB	XOR E
AC	XOR H
AD	XOR L
EE..	XOR N

Annexe 2
Complément à deux

Un octet en complément à deux se présente ainsi :

7 6 5 4 3 2 1 Ø

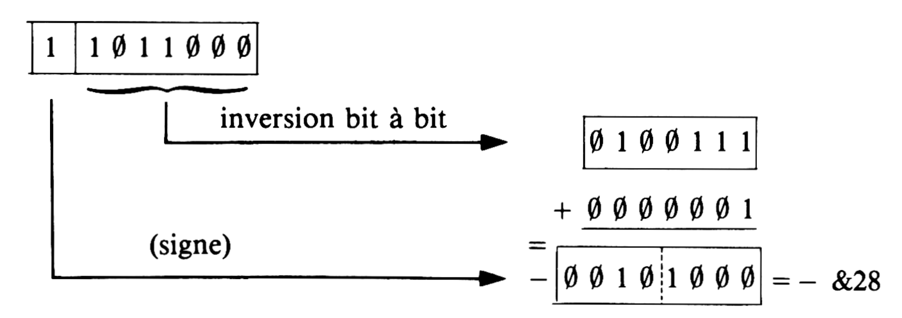
S		

Le bit 7 est le bit de signe *S*.

Si le bit de signe est Ø, le nombre est positif. Sa valeur est donnée par les bits Ø à 6.

| Ø | 1 Ø 1 | 1 Ø Ø Ø | représente + | 1 Ø 1 1 Ø Ø Ø | = + &58. |

Si le bit de signe est 1, le nombre est négatif. Sa valeur est calculée en inversant les bits Ø et 6, puis en rajoutant 1 à la valeur obtenue.



Un octet en complément à deux peut donc décrire les nombres de – 128 à + 127 :

&8Ø → – 128
&FF → – 1
&ØØ → Ø
&Ø1 → + 1
&7F → + 127

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

O

P

W

X

Y

Z

Dans la même collection

Le guide du MO5 — *André Deledicq*
Initiation au Basic TO7/TO7-70 — *Christine et François-Marie Blondel*
Un ordinateur en fête — *Serge Pouts-Lajus*
Un ordinateur et des jeux — *Jean-Pascal Duclos*
Guide pratique de l'enseignement assisté par ordinateur — *Jean-Michel Lefèvre*
Faites vos jeux en assembleur sur TO7/TO7-70 — *Michel Oury*
Jeux vidéo, jeux de demain — *Georges-Marie Becherraz/Alain Graber*
Le Basic DOS du TO7/TO7-70 et du MO5 — *Christine et François-Marie Blondel*
Basic TO7, manuel de référence — *Savema*
Initiation à Logo — Doris Avram/M. Weidenfeld/S.O.L.I.
Logo, manuel de référence — *D. Avram/T. Savatier/M. Weidenfeld/S.O.L.I.*
Logo, des ailes pour l'esprit — *Horacio C. Reggini*
Initiation au Forth — *S.E.F.I.*
Forth, manuel de référence — *S.E.F.I.*
Manuel de l'assembleur 6809 du TO7/TO7-70 — *Michel Weissgerber*
Manuel de l'assembleur 6809 du MO5 — *Michel Weissgerber*
La face cachée du TO7/TO7-70 — *Jean-Baptiste Touchard*
Manuel technique du TO7/TO7-70 — *Michel Oury*
Manuel technique du MO5 — *Michel Oury*
Macinstosh, Multiplan, Macpaint — *Eddie Adamis*
Vous et l'ordinateur APPLE — *Edward H. Carlson*
Ecrivons un programme pour APPLE — *Royal Van Horn*
Le Logo sur APPLE — *Harold Abelson*
Premiers pas avec le ZX-SPECTRUM — *Ian Stewart/Robin Jones*
Plus loin avec le ZX-SPECTRUM — *Ian Stewart/Robin Jones*
Le langage machine du ZX-SPECTRUM — *Ian Stewart/Robin Jones*
Guide pratique de l'ORIC-ATMOS du Basic à l'assembleur — *Bussac/Lagoutte*
Des programmes pour votre ORIC — *Michel Piot*
Programmer en langage machine sur Oric-Atmos et Oric-1 — *Bruce Smith*
Premiers pas avec le COMMODORE 64 — *Ian Stewart/Robin Jones*
Musique sur COMMODORE 64 — *James Vogel/Nevin B. Scrimshaw*
Programmer en assembleur sur COMMODORE 64 — *Bruce Smith*
Guide pratique du VIDEOTEX et du MINITEL — *Saboureau/Bouché*
Guide pratique de l'ordinateur personnel IBM — *Boisgontier/Dalloz/ Emery/Portefaix/Salzman*
Programmes pour MSX — *Serge Pouts-Lajus/Pierre Champeaux*
Guide pratique de l'APPLE //c — *Bruno de Latour*
Premiers pas avec l'ATARI — *Eric Deeson*
Au cœur des micros, TO7, MO5, TO7-70 — *Olivier Savin/ Jean-Claude Mariaccia*
Lire Arlequin — *Nicole Rodriguez*

Achevé d'imprimer le 3 juillet 1985 sur les presses de l'Imprimerie Carlo Descamps à Condé-sur-l'Escaut 59163. France

Dépôt légal : juillet 1985 — N° d'imprimeur : 3865

Imprimé en France

Le tour de l'Amstrad

Faites le tour de l'Amstrad pour en savoir plus sur les possibilités graphiques et sonores de son BASIC. Apprenez à vous servir des interruptions et aventurez-vous dans le langage machine pour mieux profiter des capacités de votre ordinateur. Ce livre s'adresse à tous les utilisateurs de l'Amstrad (CPC 464 et CPC 664).

ISBN : 2-7124-**0594**-3



Document numérisé avec amour par

AMSTRAD CPC

MÉMOIRE ÉCRITE



https://acpc.me/